UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA INDUSTRIAL (ESP. MECÁNICA)
DEPARTAMENTO DE INGENIERÍA MECÁNICA

PROYECTO FIN DE CARRERA

CREACIÓN DE UNA LIBRERÍA DE ELEMENTOS MECÁNICOS ROTACIONALES MEDIANTE ECOSIMPRO

Autor:
Sergio Nieva Hernando

Tutores:
Dra. Mª Jesús López Boada
Dra. Ester Olmeda Santamaría

Mayo, 2014

Contenido

CAPÍTULO 1

1. INTRODUCCIÓN

1.1.- Introducción.

Cuando alguien tiene la responsabilidad de crear o manejar un sistema cualquiera, como puede ser un sistema de suspensión para un vehículo, una planta nuclear, un banco, una ciudad, etc., debe tomar continuamente decisiones acerca de las acciones que ejecutará sobre el sistema. Estas decisiones deben ser tales que la conducta resultante del sistema satisfaga de la mejor manera posible los objetivos planteados.

Para poder decidir correctamente es necesario saber cómo responderá el sistema ante una determinada acción. Esto podría hacerse por experimentación con el sistema real pero factores como el coste o la seguridad, entre otros, hacen que esta opción generalmente no sea viable. A fin de superar estos inconvenientes, se reemplaza el sistema real por otro sistema que en la mayoría de los casos es una versión simplificada. Este último sistema es el modelo a utilizar para llevar a cabo los experimentos necesarios sin los inconvenientes planteados anteriormente. Al proceso de experimentar con un modelo se le denomina simulación.

Los ordenadores utilizan técnicas para imitar, o simular, el comportamiento de sistemas del mundo real. Para estudiar científicamente estos sistemas, a menudo se ha de hacer una serie de suposiciones acerca de cómo trabaja éste. Estas suposiciones que usualmente toman la forma de relaciones matemáticas lógicas, constituyen un modelo que va a ser usado para intentar comprender el comportamiento del sistema correspondiente. El ordenador no es la única herramienta usada en la actualidad para simular comportamientos de sistemas, hay otras tácticas que también sirven para tal fin, como pueden ser los experimentos en laboratorios a escala reducida mediante maquetas, o simuladores navales para estudiar el comportamiento de embarcaciones frente a condiciones adversas en alta mar.

En el ámbito de la ingeniería, la técnica de la simulación es usada constantemente. Se simulan toda clase de sistemas, ya sean sistemas mecánicos, eléctricos, frigoríficos, de control, etc. En este proyecto se va a utilizar una herramienta de modelado y simulación llamada EcosimPro, la cual es un potente software no causal, no es necesario escribir el orden de cálculo de las variables en las ecuaciones, que modela y simula en una dimensión (1D) cualquier tipo de sistema usando una programación declarativa. Sirve para poder experimentar con sistemas de geometrías muy simplificadas y en los cuales los componentes se intercambian flujos en 1D, pero con una gran efectividad y flexibilidad, ya que el mismo modelado sirve para hacer multitud de estudios de

simulación. EcosimPro permite modelar cualquier tipo de sistema representable por ecuaciones algebraico-diferenciales (DAE) y eventos discretos.

La metodología usada para realizar simulaciones mediante herramientas informáticas es el análisis orientado a objetos, o más precisamente, el Modelado Orientado a Objetos (MOO). El MOO es un área relativamente reciente, durante los años 90 aparecieron en la literatura algunas propuestas más elaboradas (principalmente en forma de libros) que apuntan a esta fase del desarrollo de software bajo el paradigma de la orientación a objetos [RUM99]. Existe actualmente una diversidad de técnicas, lo cual sólo aumenta la dificultad para identificar, entre otros, cuáles son las reales ventajas de abordar el análisis de sistemas con orientación a objetos, cuáles son las actividades generalmente aceptadas que deben realizarse en el MOO, y cuáles son las estrategias alternativas más apropiadas para conducir el MOO. Sin embargo, esta situación está cambiando producto del esfuerzo emprendido por la OMG (*Object Management Group*) con su propuesta UML (*Unified Modeling Language*) de estandarización notacional para los sistemas orientados a objetos [RAT99]. Más recientemente, se ha propuesto además un proceso unificado de desarrollo de software orientado a objetos [JAC99], que utiliza la notación UML, sin embargo aún está por verse si logrará transformase en estándar.

Dentro de los diferentes campos donde se puede aplicar la simulación, en el que se va a centrar el proyecto es en el campo de la ingeniería mecánica, más en concreto en el de la dinámica rotacional. Una de las características más importantes de las herramientas de simulación es la posibilidad de crear modelos de elementos independientes y poder agrupar varios de estos modelos con propiedades comunes en conjuntos, esto se conoce como crear una librería. Cuantos más componentes tenga la librería más sistemas se podrán modelar, obteniendo así una mayor flexibilidad a la hora de modelar, una mejor calidad de modelado y un mayor acierto en los resultados de la simulación.

La fiabilidad de esta técnica depende mucho de la experiencia que tenga el grupo humano que realiza la simulación. Hasta ahora no hay teoría científica que garantice la validez del proceso de simulación antes que este se realice. Para validar el modelo se ensayan alternativas conocidas y se comparan los resultados. La coincidencia de los mismos hablará de la validez del modelo para representar el sistema real. Si los resultados que el modelo arroja sobre una de esas alternativas vividas no coinciden con los reales quedará demostrada la invalidez del modelo [VOL78] [LAM91].

1.2.- Objetivos.

El objetivo del presente proyecto es la simulación de sistemas mecánicos dinámicos rotacionales mediante la herramienta de simulación EcosimPro. Para ello, se va a crear una librería con elementos mecánicos rotacionales, la cual va a contener los componentes necesarios para formar los sistemas dinámicos a simular. Una vez creados estos elementos de forma individual, se podrán unir mediante diferentes nexos para formar sistemas más complejos de manera que el estudio se pueda trasladar a situaciones reales.

Debido a la gran competencia que hay a día de hoy en todos los sectores, incluido el de la simulación, se puede encontrar en el mercado una gran variedad de herramientas para tal fin. Este proyecto se basa en una herramienta en concreto, EcosimPro, pero es recomendable simular los mismos sistemas con otras herramientas de simulación con el fin de verificar que los resultados obtenidos son correctos. El programa usado en la comparación será Matlab-Simulink [ARA06]. Matlab es un programa de gran aceptación en ingeniería destinado realizar cálculos técnicos, científicos y de propósito general. Simulink es una aplicación dentro de Matlab que permite construir y simular modelos de sistemas físicos y de control mediante diagramas de bloques, la cual será útil para discutir los resultados dados por EcosimPro y sacar conclusiones finales en el proyecto.

1.3.- Estructura.

El presente proyecto consta de seis capítulos, siendo el primero de ellos la presente introducción.

En el Capítulo 2 se destacan tres conceptos fundamentales, que constituyen el eje del proyecto. Son los conceptos de sistema, modelo y simulación. Una vez explicadas las características del modelado y de la simulación, en el Capítulo 3 se va a presentar EcosimPro. Se enumerarán las propiedades más relevantes de esta herramienta de simulación y se explicarán los conceptos fundamentales que definen este software para que el lector pueda seguir el proyecto con la mayor fluidez posible y pueda tener una clara visión de lo que se pretende con la elaboración del mismo.

Hay programas que incorporan sus propias librerías, cosa que facilita la labor a los simuladores, pero detrás de estas librerías hay mucho tiempo y esfuerzo en su creación. EcosimPro incluye una serie de librerías bastante amplia en casi todos los campos, pero en lo que a componentes mecánicos rotacionales se refiere tiene sus limitaciones, por eso, el Capítulo 4 contiene los pasos fundamentales para la creación de una librería mecánica rotacional que abarca desde elementos muy sencillos como un disco de inercia hasta elementos más complejos como pueden ser cajas de cambios.

La creación de estas librerías es fundamental para estudiar sistemas complejos formados por elementos individuales. En el Capítulo 5 del proyecto se van a simular varios sistemas mecánicos rotacionales creados a partir de los elementos modelados anteriormente en la librería. Con los conocimientos adquiridos tras el uso de EcosimPro, y para verificar que los resultados obtenidos en el proyecto son correctos, se van a simular los mismos sistemas mecánicos con otra herramienta de simulación: Matlab-Simulink comparando y analizando ambos resultados.

Para acabar, en el Capítulo 6 se indiciarán cuáles son las principales conclusiones obtenidas en el presente proyecto y por otro lado se dejarán abiertos posibles puntos de partida por si en un futuro se pretende seguir con la simulación de sistemas más complejos o de otros campos de aplicación mediante EcosimPro.

CAPÍTULO 2

2. MODELADO ORIENTADO A OBJETOS (MOO) Y SIMULACIÓN

El modelado y la simulación son práctica común a todas las disciplinas de la ingeniería y de la ciencia. Son usados en el análisis de sistemas, a fin de profundizar en su comprensión al permitir el estudio de su comportamiento aislando determinados efectos y eliminando o introduciendo perturbaciones. También son usados en el diseño de nuevos sistemas para predecir el comportamiento de los mismos antes de que sean construidos.

Los orígenes del modelado y la simulación están en la teoría de muestreo estadístico y análisis de sistemas físicos probabilísticos complejos. El aspecto común de ambos es el uso de números y muestras aleatorias para aproximar soluciones [SAT10].

Las primeras referencias sobre simulación se encuentran hacia el año 1940, cuando Von Neumann y Ullman trabajaron sobre la simulación del flujo de neutrones para la construcción de la bomba atómica en el proyecto "*Montecarlo*" [SHA88]. Desde entonces se conocían las técnicas de simulación como procesos Montecarlo, aunque en la actualidad se diferencian ambas cosas, siendo los segundos un tipo particular de simulación. También se realizó un proceso de simulación para el proyecto *"APOLLO"* dentro del plan espacial de la NASA, acerca del movimiento dentro de la atmósfera de la luna [LAW91].

Actualmente, la simulación es una poderosa técnica para la resolución de problemas por haberse producido un aumento muy significativo en el detalle y la precisión de los modelos de los procesos, en la velocidad y capacidad de memoria de los ordenadores y en la calidad del software de modelado y simulación. Los modelos matemáticos y la simulación han demostrado ser útiles tanto en la fase de investigación y desarrollo, como a la hora de realizar el diseño.

Si las relaciones que componen el modelo son suficientemente simples, es posible usar métodos matemáticos tales como el álgebra, el cálculo o la teoría de la probabilidad para obtener una información exacta de las cuestiones de interés, a esto se le llama solución analítica. Sin embargo, la mayoría de los sistemas del mundo real son demasiado complejos y normalmente los modelos realistas de los mismos, no pueden evaluarse analíticamente. Lo que sí se puede hacer es estudiar dichos modelos mediante simulación. En una simulación se utiliza el ordenador para experimentar con un modelo

numéricamente, de forma que con los resultados obtenidos se haga una estimación de las características del sistema.

2.1.- La simulación.

La simulación es el proceso por el cual se obtiene el conocimiento del comportamiento de un sistema a partir de la observación de la conducta de un modelo que lo representa. Hay que distinguir entre simulación y modelado, la simulación es una forma poco costosa y segura de experimentar con un sistema previamente modelado, pero cuyos resultados dependen de la bondad del modelo. El modelado es el proceso por el cual se crea una representación de un sistema real para su simulación.

Como se verá más adelante, gracias a la simulación se pueden reducir costes, acortar tiempo de desarrollo de productos y, minimizar materiales y riesgos en la realización de pruebas. Hacer varios diseños de un mismo producto y luego elegir la mejor variante ya no es tan complicado como lo podía ser en el pasado, incluso es posible realizar infinidad de pruebas virtuales para verificar la validez de estos diseños antes de fabricar el producto en sí. Debido a la gran versatilidad y a la elevada potencia que poseen las herramientas de simulación es posible poder modelar sistemas con alta precisión. La Figura 2.1. muestra algunas de las muchas ventajas del modelado y la simulación, entre ellas destaca el ahorro de costes y de tiempo, la posibilidad de validar o verificar el producto y las mejoras en el desarrollo del producto. Aquí todo gira en torno al dinero, desde la necesidad de crear un producto nuevo hasta la implementación del producto, pasando por el diseño del mismo. Anteriormente se ha comentado que muchas veces la mejor manera de probar un producto nuevo es someterlo a diferentes simulaciones usando un modelo previamente creado, sin necesidad de experimentar con el producto real directamente.

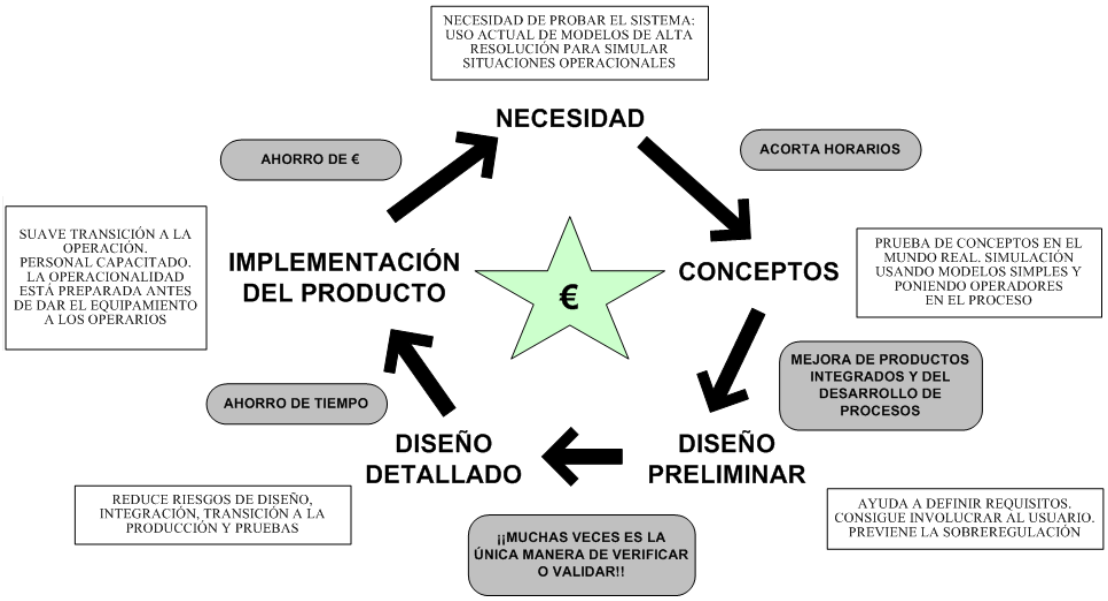


Fig. 2.1.: Ventajas del modelado y la simulación.

El poder modelar sistemas y realizar simulaciones con ellos es interesantes en ocasiones en las que el sistema físico no está disponible, cuando el experimentar el producto real

es peligroso y se pueden reducir riesgos en pruebas, cuando el coste del experimento es muy alto y se pueden ahorrar costes, cuando las constantes de tiempo del sistema son incompatibles con las del experimentador o cuando las variables de control, variables de estado o parámetros del sistema pueden no ser accesibles, etc. Con la simulación se manipulan los modelos más fácilmente, es decir, utilizando la simulación es fácil manipular los parámetros del modelo de un sistema, incluso fuera del rango admisible de un sistema físico particular. Al igual que es posible suprimir las perturbaciones y los efectos de segundo orden.

Pero la simulación también tiene una serie de inconvenientes. La facilidad de uso de la simulación es una desventaja muy importante, es sencillo para el usuario olvidar las limitaciones y condiciones bajo las que una simulación es válida, y por lo tanto sacar conclusiones erróneas de la simulación en este caso es muy probable. Para reducir estos peligros, se debería intentar siempre comparar al menos algunos de los resultados de la simulación del modelo con los resultados experimentales medidos del sistema real. Para no caer en este peligro es necesario ser consciente de que el modelo no es el sistema real, este sólo representa al sistema real bajo ciertas condiciones. No hay que forzar a que la realidad encaje dentro de las restricciones del modelo y no hay que olvidar el nivel de precisión del modelo, todos los modelos tienen hipótesis simplificadoras que hay que tener en cuenta [URQ01].

Es posible simular una gran variedad de sistemas y procesos, ya sean sistemas mecánicas, electrónicos, aeroespaciales, procesos químicos, geológicos, genéticos, etc. En la Figura 2.2. se expone un ejemplo real de modelado y simulación de un sistema de transmisión de embrague doble, en la imagen superior hay un croquis del sistema real y en la inferior una captura del modelo que representa el sistema real usando MOO.

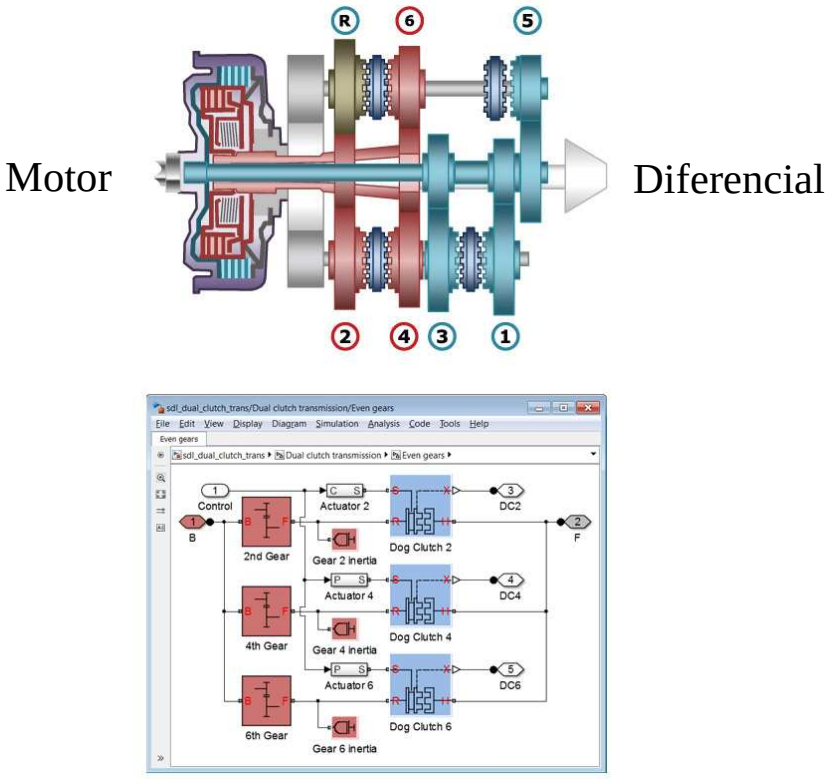


Fig. 2.2.: Ejemplo de modelado y simulación.

2.1.1.- Simulación de sistemas continuos y simulación de sistemas discretos.

En general, los modelos matemáticos de tipo dinámico representan sistemas continuos, es decir, sistemas en los que las actividades predominantes causan pequeños cambios en los atributos de sus entidades cuando las relaciones entre ellas describen las tasas o ratios de cambio de los atributos, por lo que tales modelos están definidos formalmente por ecuaciones diferenciales.

En muchos casos, a partir del modelo matemático del sistema, es posible obtener información sobre el mismo por medios analíticos, como en el caso del sistema de embrague doble que se ha expuesto en la Figura 2.2. Cuando esto no es posible, se recurre a procedimientos numéricos para resolver las ecuaciones del modelo, especialmente en el caso de los modelos dinámicos representados por ecuaciones, o sistemas de ecuaciones diferenciales, como pasa con la dinámica de poblaciones. Con el tiempo se ha ido desarrollando una amplia variedad de métodos numéricos de cálculo para resolver las ecuaciones de los modelos, una técnica numérica particular es la que se denomina simulación de sistemas, que consiste en un seguimiento a lo largo del tiempo de los cambios que tienen lugar en el modelo dinámico del sistema. En los modelos dinámicos, bien por la naturaleza del sistema modelizado, bien por las características del proceso numérico utilizado, la introducción de la aleatoriedad va a dar lugar a hablar de simulación estocástica, en este capítulo del proyecto se hará una clasificación de tipos de simulación y modelado.

La manera de efectuar el seguimiento temporal de los cambios en el modelo, que hay que suponer en correspondencia con los cambios en el sistema representado por el modelo, lleva a la aparición de dos grandes categorías dentro de la simulación de sistemas según los cambios sean continuos o discretos. En el primer caso se supone que la naturaleza del sistema permite cambios de estado continuos determinados por cambios continuos en los valores de las variables que representan el estado del sistema, mientras que en el segundo los cambios sólo pueden tener lugar en instantes discretos en el tiempo.

Para los sistemas con cambios continuos, dado que el principal interés a la hora de simular su comportamiento será reproducirlos, los sistemas de ecuaciones diferenciales serán la forma más adecuada de representarlos. Se denominará simulación continua a la simulación basada en este tipo de modelos. Los simuladores analógicos han sido muy utilizados en este tipo de simulación, aunque el desarrollo de las técnicas numéricas para la resolución de sistemas de ecuaciones diferenciales, el avance tecnológico en los ordenadores digitales y la evolución de los lenguajes de programación les han hecho perder protagonismo. Simulink, software para la simulación de sistemas dinámicos integrado en el entorno de computación numérica Matlab, el cual se usará en el Capítulo 5 para estudiar los sistemas dinámicos rotacionales creados, es un buen ejemplo de esta tendencia [MAT93].

Para los sistemas discretos, el seguimiento de los cambios de estado requiere la identificación de qué es lo que causa el cambio y cuándo lo causa, lo que se denominará un suceso. Las ecuaciones del modelo se convierten entonces en las ecuaciones y relaciones lógicas que determinan las condiciones en que tiene lugar la ocurrencia de un suceso. Un ejemplo de esta clase de sistemas sería un modelo de red de colas del taller de fabricación en el que los cambios de estado son producidos por sucesos discretos

como las llegadas de las piezas o los finales de las operaciones. Este tipo de simulación, conocida con el nombre de simulación discreta, consiste en el seguimiento de los cambios de estado del sistema que tienen lugar como consecuencia de la ocurrencia de una secuencia de sucesos [BAR96].

2.1.2.- La simulación como proceso experimental: experimentos y ordenadores.

La práctica de la simulación es una técnica que no realiza ningún intento específico para aislar las relaciones entre variables particulares, adopta un punto de vista global desde el que se intenta observar cómo cambian conjuntamente todas las variables del modelo con el tiempo. En todo caso, las relaciones entre las variables deben obtenerse a partir de tales observaciones. Esta concepción caracteriza la simulación como una técnica experimental de resolución de problemas, lo que lleva a tener la necesidad de repetir múltiples ejecuciones de la simulación para poder entender las relaciones implicadas por el sistema, en consecuencia, el uso de la simulación en un estudio debe planificarse como una serie de experimentos cuyo diseño debe seguir las normas del diseño de experimentos para que los resultados obtenidos puedan conducir a interpretaciones significativas de las relaciones de interés.

La simulación computacional es por lo tanto una técnica que realiza experimentos en un ordenador con un modelo de un sistema dado. El modelo es el vehículo utilizado para la experimentación en sustitución del sistema real. Los experimentos pueden llegar a tener un alto grado de sofisticación que requiera la utilización de técnicas estadísticas de diseño de experimentos. En la mayor parte de los casos, los experimentos de simulación son la manera de obtener respuestas a preguntas del tipo "¿qué pasaría sí?", preguntas cuyo objetivo suele ser evaluar el impacto de una posible alternativa que sirva de soporte a un proceso de toma de decisiones sobre un sistema, proceso que puede representarse esquemáticamente mediante el diagrama de la Figura 2.3. [PID92].

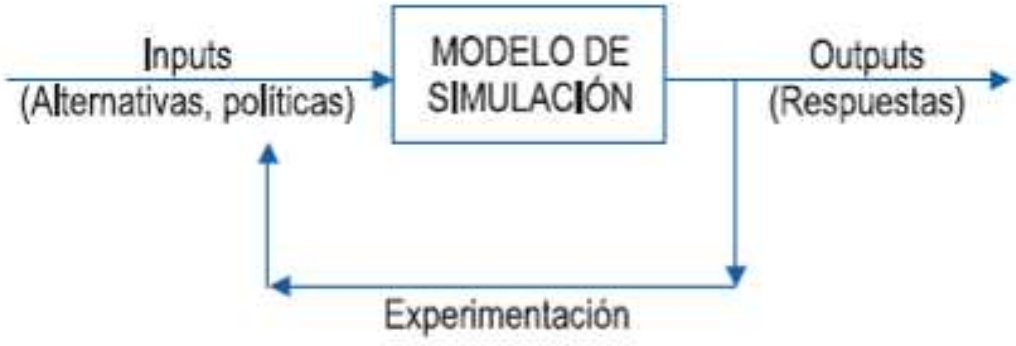


Fig. 2.3.: Esquema del proceso experimental de la simulación.

En la utilización de la simulación se pueden encontrar las características de lo que se denomina “Ingeniería de Sistemas”, es decir, una visión globalizadora que utiliza un modelo para que, al combinar elementos de análisis y diseño, se entienda por medio de experimentos cómo un sistema existente funciona o cómo puede funcionar un sistema planeado, y prever cómo las modificaciones del sistema pueden cambiar su comportamiento.

La simulación, y los experimentos de simulación, se convierten así en una herramienta de análisis de sistemas para entender cómo opera un sistema existente o cómo puede operar uno propuesto. La situación ideal en la cual el investigador realizaría los experimentos sobre el sistema real es sustituida por una en la que el investigador construye un modelo del sistema y experimenta sobre él mediante la simulación, utilizando un ordenador para investigar el comportamiento del modelo e interpretar los resultados en términos del comportamiento del sistema objeto del estudio.

La simulación y el procedimiento experimental asociado a ella, se convierten en una herramienta de diseño de sistemas cuyo objetivo es la producción de un sistema que satisfaga ciertas especificaciones. El diseñador selecciona o planea cómo son las componentes del sistema y concibe cuál debe ser la combinación de componentes y relaciones entre ellas que determinan el sistema propuesto. El diseño se traduce en un modelo cuyo comportamiento permite inducir el del sistema previsto, este se acepta cuando las previsiones se ajustan adecuadamente a los comportamientos deseados, en caso contrario se introducen las modificaciones pertinentes en el modelo y se repite el proceso.

Otra posibilidad es la que se da en estudios económicos, políticos, médicos, etc., en los que se conoce el comportamiento del sistema pero no los procesos que producen tal comportamiento. En este caso se formulan hipótesis sobre las entidades y actividades que pueden explicar la conducta. El estudio de simulación por medio del modelo correspondiente permite comparar las respuestas de un modelo basado en tales hipótesis con el comportamiento conocido, de manera que una concordancia adecuada lleva a suponer que la estructura del modelo se corresponde con la del sistema real.

La aplicación de la simulación a diferentes tipos de sistemas combinada con las diferentes clases de estudio que se pueden realizar conduce a una gran cantidad de variantes de la manera en que se puede realizar un estudio de simulación. Sin embargo, hay determinados pasos básicos del proceso que pueden identificarse como los constituyentes de lo que se denominará la metodología de un estudio de simulación, estos pasos son los siguientes:

1. Definición del problema y planificación del estudio.
2. Recogida de datos.
3. Formulación del modelo matemático.
4. Construcción y verificación del programa para ordenador del modelo.
5. Ejecuciones de prueba del modelo.
6. Validación del modelo.
7. Diseño de los experimentos de simulación.
8. Ejecución de los experimentos.
9. Análisis de los resultados.

El proceso generalmente no es secuencial, sino iterativo. Puede que se tenga que repetir algunos de los pasos que hay que dar en función de los resultados intermedios obtenidos, esto se puede ver en la Figura 2.4.

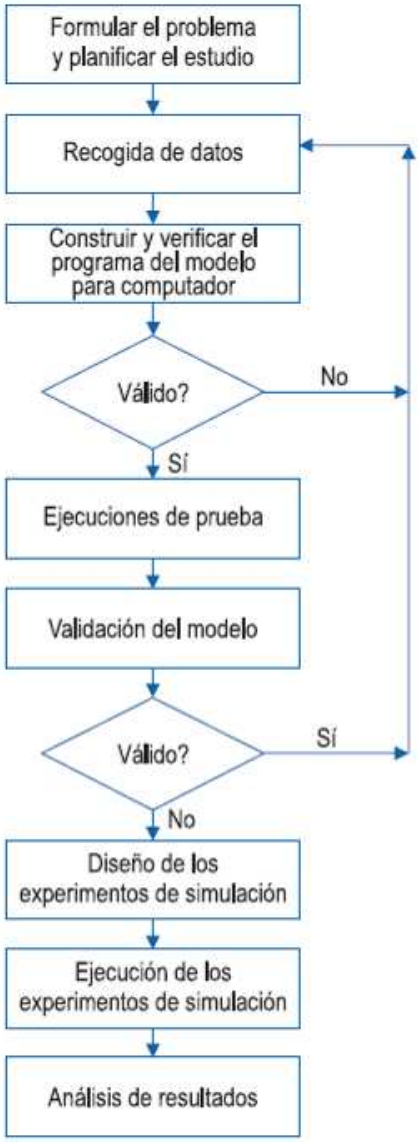


Fig. 2.4.: Etapas de un estudio de simulación.

Ningún estudio de simulación puede llevarse a cabo sin establecer claramente una definición precisa del problema que se pretende resolver y los objetivos del estudio. Los diseños alternativos del sistema que se han de estudiar, han de quedar claramente especificados así como los criterios para evaluar dichos diseños, estos criterios servirán de base al proceso de toma de decisiones para elegir uno de los diseños. Para la formulación del modelo debe establecerse su estructura definiendo cuáles son los aspectos del funcionamiento del sistema que son significativos para la resolución del problema que se tiene entre manos y qué datos son necesarios recoger para proporcionar al modelo la información adecuada.

La construcción del modelo de simulación es en muchos casos más un arte que una ciencia, que combina aspectos matemáticos y lógicos. En general, la experiencia recomienda empezar con modelos moderadamente detallados que paulatinamente se van haciendo más sofisticados. El modelo únicamente debe contener el nivel de detalle requerido por los objetivos del estudio. Dado un modelo matemático, la construcción del programa para ordenador es el requisito imprescindible para poder manipular

numéricamente el modelo y así obtener las soluciones que respondan a las preguntas que el analista se formula sobre el sistema.

La validación del modelo es uno de los pasos cruciales del proceso, suele ser uno de los más difíciles, pero es un requisito indispensable para establecer si el modelo representa o no adecuadamente el sistema objeto del estudio, de manera que se puedan garantizar las inducciones y extrapolaciones sobre el comportamiento del sistema a partir de lo observado sobre el modelo.

Diseñar los experimentos significa, como se ha comentado anteriormente, aplicar rigurosamente las técnicas observacionales de la estadística, propias del método científico, que permitan garantizar la significación de las respuestas producidas por la ejecución del programa que implanta el modelo en el ordenador [BAR96].

2.1.3.- Modelos de simulación frente a soluciones analíticas.

Todos los modelos pueden ser resueltos de forma exacta, unos mediante ecuaciones diferenciales que modelizan el sistema (tal es el caso del doble embrague propuesto como ejemplo en la Figura 2.2.), otros mediante el uso de métodos numéricos con las ecuaciones que definen la dinámica de poblaciones, y los últimos mediante métodos numéricos aproximados para los modelos de tráfico.

Estos procedimientos constituyen el primer atisbo de lo que se ha denominado procedimientos de simulación como alternativa a los métodos analíticos. La pregunta es en qué van a consistir y cuándo hay que aplicar lo que propiamente se denomina simulación en el caso de los sistemas discretos.

Una modelización analítica de un sistema puede presentar más o menos dificultades dependiendo de la complicación del propio sistema. Para hacerse cargo de estas dificultades puede dividirse el sistema en subsistemas más simples, para los que la solución analítica de equilibrio es más sencilla. Sin embargo, bastaría que para este caso se preguntase por el comportamiento durante el período transitorio para que la situación, aun siendo tratable analíticamente, se complique bastante. Algunas veces las preguntas relativas al estado estacionario o de equilibrio son aquellas que suponen el funcionamiento del sistema a largo plazo, mientras que en otras ocasiones lo que interesa es analizar el proceso de arranque del sistema, es decir, lo que se denomina el estado transitorio antes de entrar en la supuesta condición de equilibrio a largo plazo.

Para responder a tal pregunta sobre el sistema se tendría que encontrar la solución al sistema siguiente de ecuaciones diferenciales, Ecuaciones (2.1.) y (2.2.), (no hay que olvidar que se trata de un sistema dinámico pero cuyos cambios de estado son discretos):

$$\frac{dP_k(t)}{dt} = -(\lambda+\mu)P_k(t) + \lambda P_{k-1}(t) + \mu P_{k+1}(t),\ k \geq 1 \qquad (2.1.)$$

$$\frac{dP_0(t)}{dt} = -\lambda P_0(t) + \mu P_1(t), \quad k = 0 \tag{2.2.}$$

donde $P_k(t)$ es la probabilidad de que el sistema se encuentre en el estado *k* en el instante *t*. La distribución de probabilidad de estados función del tiempo viene dada por Kleinrock [KLE75] en el Ecuación (2.3.):

$$P_k(t) = e^{-(\lambda+\mu)t}\left[\rho^{(k-1)/2} I_{k-1}(at) + \rho^{(k-i-1)/2} I_{k+i+1}(at) + (1-\rho)\rho^k \sum_{j=k+1+2}^{\infty} \rho^{-j/2} I_j(at)\right] \tag{2.3.}$$

donde

$$a = 2\mu\rho^{1/2} \tag{2.4.}$$

$$I_k(x) = \sum_{m=0}^{\infty} \frac{(x/2)^{k+2m}}{(k+m)!\,m!}, \quad k \geq -1 \tag{2.5.}$$

que es la función de Bessel modificada de primera clase de orden *k*.

Complicaciones analíticas similares, o de orden superior, aparecen cuando las hipótesis sobre las distribuciones de probabilidad dejan de ser poissonianas o exponenciales para ser simplemente normales o de Erlang, como corresponde a muchas situaciones reales. Estas dificultades inherentes a las soluciones analíticas pueden ser solventadas con relativa facilidad por medio de la simulación, para obtener soluciones numéricas aproximadas.

A pesar de su utilidad, la simulación no puede considerarse como una panacea capaz de resolver todo tipo de situaciones, aun contando con la ayuda de los lenguajes especializados para la simulación, tal es el caso de EcosimPro Language (el lenguaje que usa EcosimPro) y Modelica, de los que se hablará en el Capítulo 3, o con los avances que han representado los entornos software para simulación. La realización de un estudio de simulación puede dar lugar a un esfuerzo y un consumo de recursos no despreciable en cualquiera de sus fases: definición del problema, recogida de información, construcción del modelo y programación del mismo o realización de los experimentos de simulación en ordenador. Especialmente en este último caso, sistemas complejos pueden conducir a programas largos y complicados que requieran cantidades importantes de recursos computacionales. Estas han sido algunas de las razones por las que en ciertos dominios de aplicación la simulación ha sido considerada como un último recurso al que acudir cuando todo lo demás falla.

Sin embargo, por sus características y por los desarrollos computacionales que se han conseguido en los últimos años, la simulación sigue presentando una serie de ventajas

que no sólo la convierten en el procedimiento más adecuado en muchos casos, sino que hacen que sea la única alternativa tecnológica.

Esto resulta especialmente obvio en aquellos casos en los que las características del sistema que se pretende estudiar hacen inviable, por razones físicas o de coste, la experimentación directa sobre el sistema. El mundo de la producción industrial, de la aeronáutica, de la industria del automóvil, del tráfico, etc., son claros ejemplos de esta situación en la que, si bien es cierto que en algunos casos se puede recurrir a modelos analíticos, también lo es que tales modelos no siempre son capaces de recoger todos los aspectos de interés del sistema que conducirían a modelos inviables o para los que no se dispone de herramientas adecuadas, obligando a introducir una serie de hipótesis simplificadoras que pueden resultar inadecuadas en función de los objetivos del estudio. Las soluciones para los períodos transitorios pueden ser complicadas de obtener analíticamente en contraste con las simplicidades de los procedimientos para obtener las soluciones estacionarias, sin embargo, basta introducir hipótesis adicionales aparentemente sencillas, que aproximan el modelo a otras situaciones reales, para entrar rápidamente en el terreno de las dificultades analíticas crecientes. Los modelos con distribuciones de probabilidad de llegadas y servicios de tipo general, inclusión de impaciencias o políticas basadas en prioridades, etc., son un buen ejemplo de ello.

Incluso en aquellos casos en los que es posible la experimentación directa, la simulación puede ofrecer ventajas tales como un coste inferior, tiempo, repeticiones y seguridad. Aun siendo viables los experimentos directos con el sistema físico, pueden con frecuencia tener un coste muy superior al de la simulación a pesar de los esfuerzos para construir el modelo, y el tiempo y recursos computacionales requeridos para la ejecución de los experimentos.

Aunque el desarrollo de un modelo adecuado, y su programación para ser ejecutado en un ordenador, puede requerir una cantidad de tiempo significativa, una vez construido y depurado el modelo de simulación representa una atractiva posibilidad para trabajar con las más variadas escalas de tiempo, minutos, horas, semanas, meses, años, etc. En unos pocos segundos de tiempo de ordenador se pueden comparar colecciones variadas de alternativas a través de experimentos de simulación que siempre pueden repetirse en las más diversas condiciones, lo que no siempre es posible en los experimentos con el sistema real, basta pensar en sistemas de fabricación, gestión de empresas, etc.

Es frecuente que los experimentos persigan el objetivo de determinar la respuesta del sistema en condiciones extremas, lo que puede resultar peligroso o incluso ilegal en la vida real. Las aplicaciones de la simulación en aeronáutica, o en la gestión de aeropuertos, constituyen buenos ejemplos de lo que se quiere significar.

El dilema modelos analíticos frente a modelos de simulación debe resolverse en cada caso ateniéndose al tiempo de sistema, los objetivos del estudio, las características del modelo, los costes, etc. La cuestión clave es la habilidad y capacidad para construir el modelo del sistema, si este es analítico y las hipótesis de modelización no obligan a simplificaciones que invaliden la capacidad del modelo para responder a las cuestiones de interés que se planteen sobre el sistema, entonces las soluciones analíticas del modelo matemático pueden ser suficientes.

Si el conocimiento del sistema no permite formular hipótesis que conduzcan a una completa formalización del modelo en términos analíticos, o el requisito de no realizar hipótesis simplificadoras conduce a modelos matemáticos de difícil o imposible tratamiento matemático, entonces posiblemente la simulación será la alternativa válida si no es la única posible.

El analista del sistema no debe olvidar que un mismo sistema puede representarse formalmente mediante diversos modelos en función de los problemas que el analista se plantea sobre el sistema. Law y Kelton formalizaron el proceso de decisión conocido como sistema matemático-modelo de simulación, en el cual se puede experimentar sobre el sistema real o sobre un modelo del sistema. En el diagrama de la Figura 2.5. se observan las diferentes formas de estudiar un sistema según Law y Kelton [KEL91].

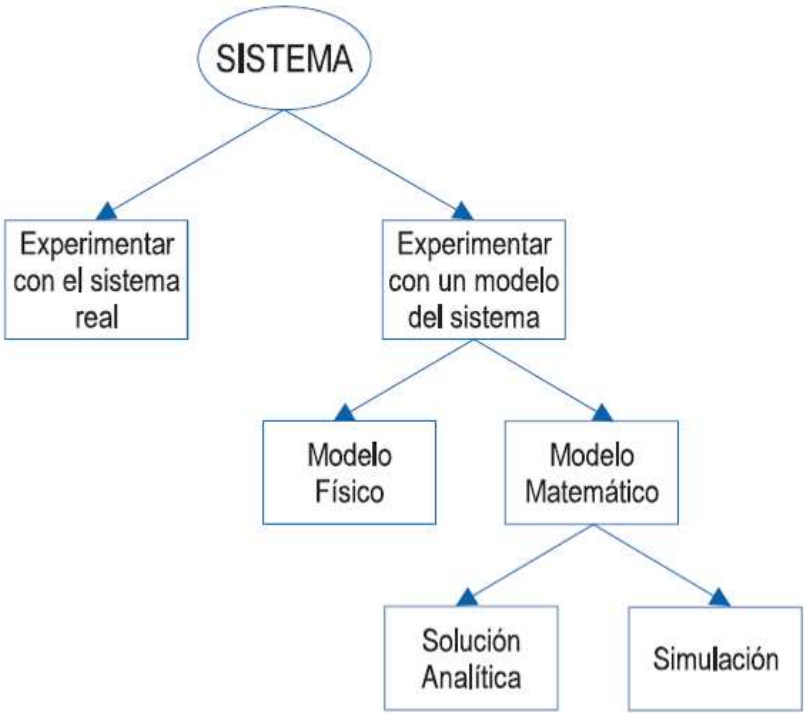


Fig. 2.5.: Maneras de estudiar un sistema según Law y Kelton.

El desarrollo experimentado por el software de simulación en estos últimos años ha hecho más fácil el uso de la simulación, lo que ha incrementado notablemente su uso frente al de otros métodos para estudiar sistemas. De lo expuesto hasta aquí se desprende claramente que si bien la simulación tiene muchas ventajas no deja de presentar algunos problemas, especialmente cuando se usa indebidamente, que cabe tener en cuenta para paliarlos o, si es posible, evitarlos, pues de lo contrario pueden invalidar los resultados de un proyecto de simulación. Law y Kelton resumen en su texto la situación de la manera siguiente [KEL91].

La simulación es recomendable, o incluso puede ser la única alternativa posible, para investigar sistemas complejos en los que estén presentes elementos estocásticos que difícilmente pueden ser tratados con la precisión adecuada en un modelo matemático. La simulación permite, con relativa facilidad, estimar el funcionamiento del sistema bajo condiciones de operación alternativas, es la herramienta para comparar diseños alternativos de un sistema que tengan que satisfacer requerimientos específicos. La

simulación permite mantener un mayor control sobre las condiciones experimentales que el que se puede mantener en la experimentación con el sistema físico. Por otra parte, la simulación permite estudiar el comportamiento del sistema en períodos de tiempo de cualquier longitud, comprimidos a la duración de la ejecución del simulador en un ordenador.

Sin embargo, no se debe olvidar que la simulación de un modelo estocástico únicamente produce estimaciones de las características verdaderas del modelo para un conjunto particular de parámetros de entrada, lo que implica la necesidad de diseñar adecuadamente los experimentos de simulación y repetirlos en cantidad suficiente como para garantizar la calidad de las estimaciones. De ahí que los modelos de simulación no sean tan buenos cuando se pretende optimizar el rendimiento como lo son cuando lo que se persigue es comparar entre sí diseños alternativos. En particular, para sistemas para los que puede definirse un modelo analítico adecuado que proporciona resultados exactos, este será preferible al modelo de simulación. No hay que olvidar en ningún caso que si un modelo no es una representación válida del sistema que se estudia la información que proporcione será de poca utilidad.

Modelos de simulación y modelos analíticos no deben considerarse siempre como antitéticos, en muchos casos pueden jugar un papel complementario sirviendo la simulación para verificar la validez de las hipótesis para el modelo analítico, o el modelo analítico para sugerir cuales son las alternativas razonables que hay que investigar por simulación.

Finalmente, suponiendo que se ha decidido proceder a un estudio de simulación, no hay que olvidar las recomendaciones metodológicas que, de no cumplirse estas, pueden llevar al fracaso del estudio, como pasa por ejemplo cuando no están bien definidos los objetivos del estudio, se construye un modelo con un nivel de detalle inadecuado, se concibe el estudio de simulación como una especie de ejercicio de programación de computadores, se utiliza inadecuadamente las capacidades de animación de resultados, no se identifican correctamente las fuentes de aleatoriedad del sistema, se utilizan distribuciones de probabilidad que no se corresponden como es debido con dichas fuentes de aleatoriedad, se utilizan métodos estadísticos incorrectos para analizar los resultados (por ejemplo los que suponen independencia), o se intenta extraer conclusiones de una única ejecución del simulador, etc.

2.1.4.- La simulación de los sistemas discretos.

En los desarrollos anteriores han sido analizadas las limitaciones que tiene el tratamiento analítico para manipular numéricamente los modelos matemáticos que representan los sistemas que interesa estudiar. Limitaciones que se hacen más patentes cuando el modelo no es puramente analítico sino que incluye, como parte de sus reglas de operación, condiciones de tipo lógico. Se pone de manifiesto cómo es posible llegar rápidamente a grandes complejidades analíticas con modelos de sistemas de características aleatorias. En estos casos se ha aludido a la simulación como una técnica numérica que permite superar estas limitaciones e inconvenientes, pero, ¿en qué consiste esta técnica? Para el caso de los modelos continuos los procedimientos de resolución numérica de las ecuaciones diferenciales del modelo son la componente

fundamental de lo que se entiende por simulación continua. La simulación de los sistemas discretos requiere un tratamiento diferente.

De acuerdo con los planteamientos expuestos, una posibilidad consiste en considerar que el sistema evoluciona en el tiempo cambiando de estado, y que simular consiste en ser capaz de imitar los cambios de estado del sistema emulando su evolución. Ello requiere definir qué se entiende por estado del sistema y cómo cambia de estado el sistema.

Se puede definir el estado del sistema, por ejemplo, como el número de unidades que hay en el sistema en un momento dado. Se dirá pues que el sistema se encuentra en el estado *n* en el instante *t* si en dicho instante hay *n* unidades en el sistema. A partir de esta especificación del estado del sistema, es evidente que el sistema cambia de estado cada vez que llega a él una nueva unidad o cada vez que una unidad lo abandona. Se dirá entonces que el sistema cambia de estado cada vez que se produce un suceso y que hay dos tipos de suceso susceptibles de cambiar el estado del sistema, el suceso llegada de una nueva unidad al sistema y el suceso salida de una unidad del sistema.

Si las llegadas son aleatorias según una distribución de Poisson, y las salidas son exponenciales, simular sucesos llegada al sistema consiste en extraer muestras de números aleatorios que sigan la distribución de Poisson que las describe y simular sucesos salida consiste en extraer muestras de números aleatorios que sigan la distribución exponencial que los describe.

La simulación permite recoger observaciones sobre los estados del sistema y de esta manera estimar los parámetros que se necesita conocer. En caso de que las distribuciones de llegada o salida no fuesen poisonianas o exponenciales, sino de cualquier otro tipo, normal, Erlang, etc., el proceso metodológico de la simulación sería el mismo, la única variación estribaría en la extracción de muestras aleatorias de las distribuciones de probabilidad correspondientes [BAR96].

2.2.- Modelado Orientado a Objetos (MOO).

Para poder simular un sistema es necesario crear un modelo que lo represente, para ello es necesario recurrir a una de las dos fuentes de conocimiento de las propiedades de los sistemas. Por una parte está la recopilación de la experiencia de los usuarios y de la literatura del área en cuestión, y por otra parte está el sistema en sí. Hay dos caminos para construir modelos: el modelado físico, que consiste en descomponer el sistema en subsistemas con propiedades conocidas, o la identificación del sistema, es decir, analizar los sistemas y construir el modelo con las propiedades que describen el sistema.

Para el modelado de sistemas por ordenador es conveniente usar la metodología de Modelado Orientado a Objetos (MOO). El lenguaje de MOO usa los objetos y sus interacciones para diseñar aplicaciones informáticas y llevar a cabo simulaciones con ellas. Está basado en varias técnicas, incluyendo herencia, abstracción, polimorfismo y encapsulamiento de las cuales se hablará más adelante en este capítulo.

2.2.1.- El proceso de construcción de modelos: modelos matemáticos.

El análisis del sistema a través de un modelo implica que la representación del sistema que constituye el modelo ha de ser una representación manipulable numéricamente. El ejercicio de construcción del modelo del sistema comienza por la construcción de un modelo conceptual del sistema, representación equivalente lógica aproximada del sistema real que, como tal, constituye una abstracción simplificada del mismo y que a continuación se traduce en un modelo apto para su ejecución en un ordenador. El proceso de modelización o construcción del modelo implica:

- Identificación de las entidades principales del sistema y de sus atributos característicos.
- Identificación y representación de las reglas que gobiernan el sistema que se quiere simular.
- Captación de la naturaleza de las interacciones lógicas del sistema que se modeliza.
- Verificación de que las reglas incorporadas al modelo son una representación válida de las del sistema que se modeliza.
- Representación del comportamiento aleatorio.

Una precaución importante a tener en cuenta cuando se construye un modelo es que ningún modelo es mejor que las hipótesis que encierra. Traducir el modelo a un modelo específico para ordenador consiste en representar el modelo conceptual mediante un lenguaje apto para su ejecución en un ordenador. Este proceso se simplifica cuando la representación se hace utilizando un lenguaje especializado orientado a problemas específicos. Las etapas del proceso de construcción del modelo se sintetizan en la Figura 2.6.

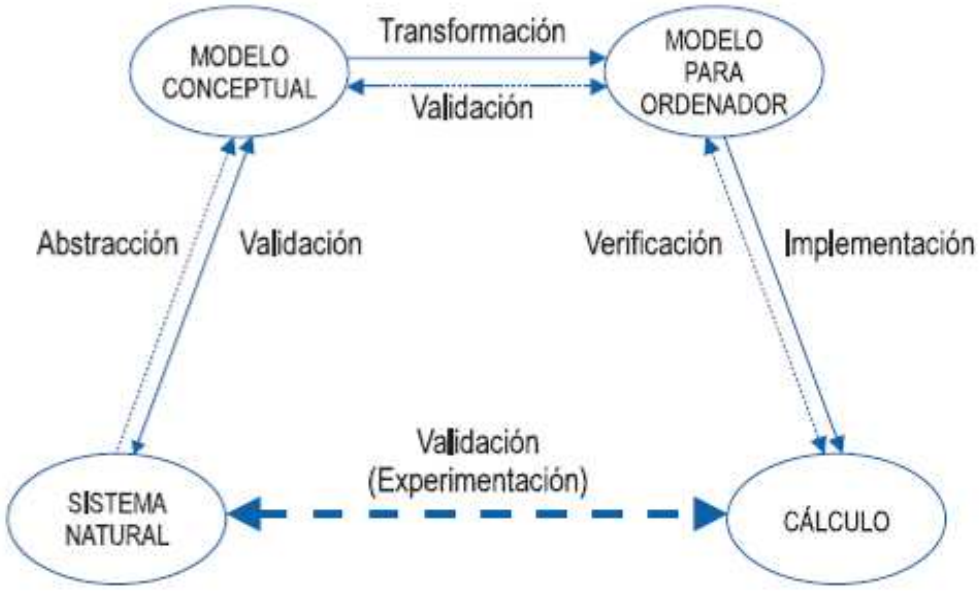


Fig. 2.6.: Proceso de modelización.

2.2.2.- Tipos de modelos.

Se puede hacer una primera clasificación de modelos dependiendo del tiempo en el que puedan cambiar sus variables:

- Modelos de tiempo continuo: se caracterizan porque en intervalos finitos de tiempo, las variables de estado cambian sus valores infinitas veces. Los modelos de este tipo están representados por conjuntos de ecuaciones diferenciales.

- Modelos de tiempo discreto: en este tipo de modelos las variables sólo pueden cambiar en determinados instantes de tiempo, por tanto, el eje temporal se discretiza.

- Modelos híbridos: combinan los dos modelos anteriores. Aquí se incluyen los modelos de sistemas dinámicos con variables de estado continuas y discretas.

- Modelos cualitativos: son modelos discretos aunque con la discretización del eje temporal no equidistante.

- Modelos de sucesos discretos: el eje temporal y el eje de estado son continuos, pero sólo puede ocurrir un número finito de cambios.

Se deben proporcionar sentencias que permitan modelar los dos mundos, el continuo y el discreto. Con el modelado continuo, el usuario expresa el modelo físico con unas ecuaciones matemáticas concretas, ya sean algebraicas o diferenciales con respecto al tiempo. Pero no sólo se debe permitir una formulación continua, cualquier modelo físico que se construya con ecuaciones continuas en algún instante tendrá un comportamiento discreto. Cada tipo de comportamiento continuo está asociado a una serie de eventos que se están comprobando continuamente. Cuando ocurre uno de ellos se ejecuta una serie de acciones asociadas. Esto significa que los componentes se modelan utilizando ambos métodos de modelado, el discreto y el continuo.

Existen otras diferentes clasificaciones de modelos matemáticos atendiendo a diferentes criterios, entre ellos [VAZ10]:

- Modelos estocásticos o deterministas. Se clasifican así en función de que trabajen o no con variables aleatorias y procesos estocásticos.

- Modelos dinámicos o estáticos. Si en el modelo existe una ligadura directa e instantánea entre las variables, se dice que es estático. Por el contrario, los sistemas cuyas variables pueden cambiar sin necesidad de que exista una influencia directa externa se denominan dinámicos.

- Modelos de parámetros agrupados o de parámetros distribuidos. Los modelos de parámetros agrupados están expresados mediante ecuaciones diferenciales ordinarias y los modelos de parámetros distribuidos están descritos mediante ecuaciones diferenciales en derivadas parciales.

2.2.3.- Lenguaje de MOO.

En 1967, el lenguaje de programación Simula aplicaba algunas ideas para modelar aspectos de la realidad de forma mucho más directa que los métodos tradicionales [CAM00]. Desde entonces, la orientación a objetos (OO) ha adquirido cada vez mayor popularidad al demostrar sus ventajas, alguna de las cuales son:

- Permite un modelado más "natural" de la realidad.
- Facilita la reutilización de componentes de software.
- Ofrece mecanismos de abstracción para mantener controlable la construcción de sistemas complejos.

En el mercado aparecen constantemente herramientas y lenguajes de programación autodenominados orientados a objetos y los ya existentes evolucionan rápidamente incluyendo nuevas características de OO. De la misma manera, se han desarrollado múltiples métodos y metodologías bajo este enfoque, cada una con aportaciones propias que llegan, en ocasiones, a resultar contradictorias entre sí. Como ya se ha comentado en la introducción, se ha logrado la creación de un lenguaje unificado para el modelado, llamado UML (*unified modeling language*) [RAT99]. La intención de UML es ser independiente de cualquier metodología y es precisamente esta independencia la que lo hace importante para la comunicación entre desarrolladores, ya que las metodologías son muchas y están en constante evolución.

Por desgracia, a pesar de muchos esfuerzos y de importantes avances, las ciencias de la computación no han creado aún una definición de modelo de objetos como tal. En un panorama como éste, es indispensable la existencia, de al menos, un modelo informal de objetos que oriente la evolución de la tecnología y que tenga la aprobación de los expertos en la materia. Un modelo así permitiría su estudio consistente por parte de los profesionales de las tecnologías de la información, facilitaría la creación de mejores lenguajes y herramientas y, lo que es más importante, definiría los estándares para una metodología de desarrollo consistente y aplicable.

Sin embargo, este modelo no existe, lo que provoca inconsistencias incluso en el tratamiento de los principios y conceptos básicos de la OO. Por eso, es frecuente encontrar errores graves en el desarrollo de sistemas OO y, lo que es aún peor, se implementan soluciones de dudosa validez en herramientas de desarrollo que se dicen orientadas a objetos.

Aun sin haber alcanzado la madurez, la orientación a objetos es el paradigma que mejor permite solucionar los muchos y variados problemas que existen en el desarrollo de software.

Este tipo de lenguaje posee inteligencia propia para evitar tener que escribir el modelo mediante un conjunto de ecuaciones con la causalidad asignada de antemano. Al modelar elementos no es necesario escribir el orden de cálculo de las variables de las ecuaciones, se usa una programación declarativa en la que se pueden hacer varios experimentos con un mismo modelado, esto se conoce como herramienta de simulación no-causal. El núcleo de la herramienta se encarga de ordenar y reescribir las ecuaciones

de la forma adecuada, eliminando las variables redundantes y las ecuaciones triviales. Algunos conceptos y propiedades importantes del MOO son las siguientes:

- **Componente (objeto):** elemento fundamental del MOO. Un componente reúne toda la información sobre el comportamiento dinámico, tanto continuo como discreto, de todo el sistema a modelar. Un componente puede ser tan simple como una ecuación diferencial o ser tan complejo como una planta desaladora.

- **Clase:** definiciones de las propiedades y comportamientos de un componente concreto.

- **Puertos:** conexiones entre componentes. La definición de un puerto incluye las variables de conexión y sus comportamientos, y las restricciones cuando se conectan a más de un puerto.

- **Instanciación:** es la lectura de las clases y la creación de objetos a partir de ellas.

- **Modularidad:** propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos) fáciles de manejar, cada una de las cuales debe ser tan independiente de la aplicación en sí y de las restantes partes como sea posible. Estos módulos tienen conexiones con otros módulos y se pueden compilar por separado, entendiéndose compilar como el proceso de traducción de un código fuente escrito en un lenguaje de programación a lenguaje máquina para que pueda ser ejecutado por el ordenador. El desarrollo modular permite a un sistema ser modelado desde abajo hacia arriba, es decir, programando primero los componentes más básicos para, a partir de ellos, ir desarrollando otros más complejos.

- **Herencia:** normalmente las clases no están aisladas, sino que se relacionan entre sí formando una jerarquía de clasificación. Los componentes heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. La herencia organiza y facilita el polimorfismo y el encapsulamiento permitiendo a los componentes ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir y extender su comportamiento sin tener que volver a implementarlo. Esto suele hacerse habitualmente agrupando los componentes en clases y estas en árboles o enrejados que reflejan un comportamiento común. Cuando un componente hereda de más de una clase se dice que hay herencia múltiple.

- **Componente abstracto (abstracción):** componente que no tiene significado real y se utiliza sólo para unir un conjunto de variables, ecuaciones y eventos discretos que pueden ser heredados por otros.

- **Encapsulamiento:** significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión de los componentes del sistema, en otras palabras, consiste en separar los aspectos externos de los objetos, a los cuales pueden acceder otros objetos, de los detalles internos de la implementación del mismo, que quedan ocultos a los demás.

- **Polimorfismo:** concepto ligado a la reutilización de modelos. Comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre, al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando. O dicho de otro modo, las referencias y las colecciones de componentes pueden contener componentes de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del componente referenciado.

- **Agregación:** es una forma fuerte de asociación, en la cual el objeto está formado por componentes previamente creados. En los lenguajes MOO todos los elementos de modelado son componentes. Un componente puede heredar de otro y puede tener múltiples instancias, esto refuerza la característica de reutilización ya que permite que los componentes sean creados a partir de otros.

Además de las características que se han mencionado anteriormente como esenciales de los lenguajes de programación orientados a objetos, es deseable que éstos cumplan también con las siguientes propiedades:

- Tipificación fuerte. Esto es, que durante la fase de diseño e implementación se declare que tipo de datos soportara cada variable.

- Manejo de excepciones. Dentro de la misma definición del lenguaje se deberá establecer la forma de detectar y manipular excepciones que puedan surgir durante la ejecución de un programa.

- Paso de mensajes. Es conveniente que el lenguaje soporte paso de mensajes entre módulos de manera bidireccional.

- Generalidad. Se refiere principalmente a que las clases se definan lo más generalizadas posible para que sean fácilmente reusables. Para generar este tipo de clases normalmente se definen parámetros formales que son instanciados por parámetros reales.

- Multitarea. Es conveniente que el lenguaje permita la creación de procesos que se ejecuten de forma simultánea, independientemente del sistema operativo.

- Persistencia. Los objetos deben poder permanecer, si así se desea, después de la ejecución de un programa.

- Datos compartidos. Los objetos pueden necesitar referirse a la misma localidad de memoria (memoria compartida), o bien comunicarse mediante mensajes.

2.2.4.- Taxonomía de los lenguajes MOO.

Wegner creó una clasificación para los lenguajes de programación con respecto a la orientación a objetos en 1987 [JOY98], dicha clasificación se constituye de la siguiente forma:

- Basados en objetos: si la sintaxis y semántica de lenguaje soportan las características de objetos que se han mencionado.

- Basados en clases: si un lenguaje está basado en objetos y además soporta clases, se dice que está basado en clases.

- Orientación a objetos: si un lenguaje de programación soporta objetos, clases y además permite la jerarquía de dichas clases, entonces se dice que es un lenguaje de programación orientado a objetos.

Esta clasificación es la que prevalece actualmente, en esta se establece claramente que para que un lenguaje de programación sea considerado orientado a objetos éste debe soportar la creación de clases y la herencia.

Java es actualmente uno de los lenguajes de programación que cumple perfectamente con los requisitos de la orientación a objetos. Delphi, Simula o Smalltalk son también considerados lenguajes orientados a objetos.

Java puede ser visto como el resultado de una mezcla de C++ y Smalltalk, tiene la sintaxis de C++, lo que lo hace relativamente fácil de aprender (si es que se está familiarizado con C++). Una de las características que lo hace más robusto que C++ es que no tiene apuntadores, que son causa común de errores en el desarrollo de aplicaciones con este lenguaje. Como en el caso de Smalltalk, Java tiene recolección de "basura", una capacidad que libera al programador de definir funciones de alojamiento y desalojamiento de estructuras en memoria [STR97].

2.2.5.- Forma de trabajar con MOO.

La forma de trabajar con lenguajes MOO es, normalmente, la que se presenta en la Figura 2.7. En primer lugar hay que generar los modelos de los componentes (objetos) y puertos (uniones entre objetos) creando para ello sus códigos en el lenguaje de programación. Es conveniente organizar estos componentes en una librería para tener todo mucho más ordenado, incluso es posible asignar iconos gráficos a estos componentes para tener mayor claridad a la hora de trabajar con ellos (la potencia de un entorno está en el número y en la calidad de sus librerías). Una vez creada la librería con la que se va a trabajar es imprescindible generar las ecuaciones que representan la mecánica del sistema. Creadas estas ecuaciones que definen el modelo se pueden ordenar de forma secuencial, definir qué incógnitas se van a despejar, decir cómo se resuelven los posibles lazos algebraicos, etc., pero esto es algo que algunos de los programas de modelado y simulación hacen automáticamente. Finalmente hay que ejecutar las simulaciones y los experimentos que se deseen.

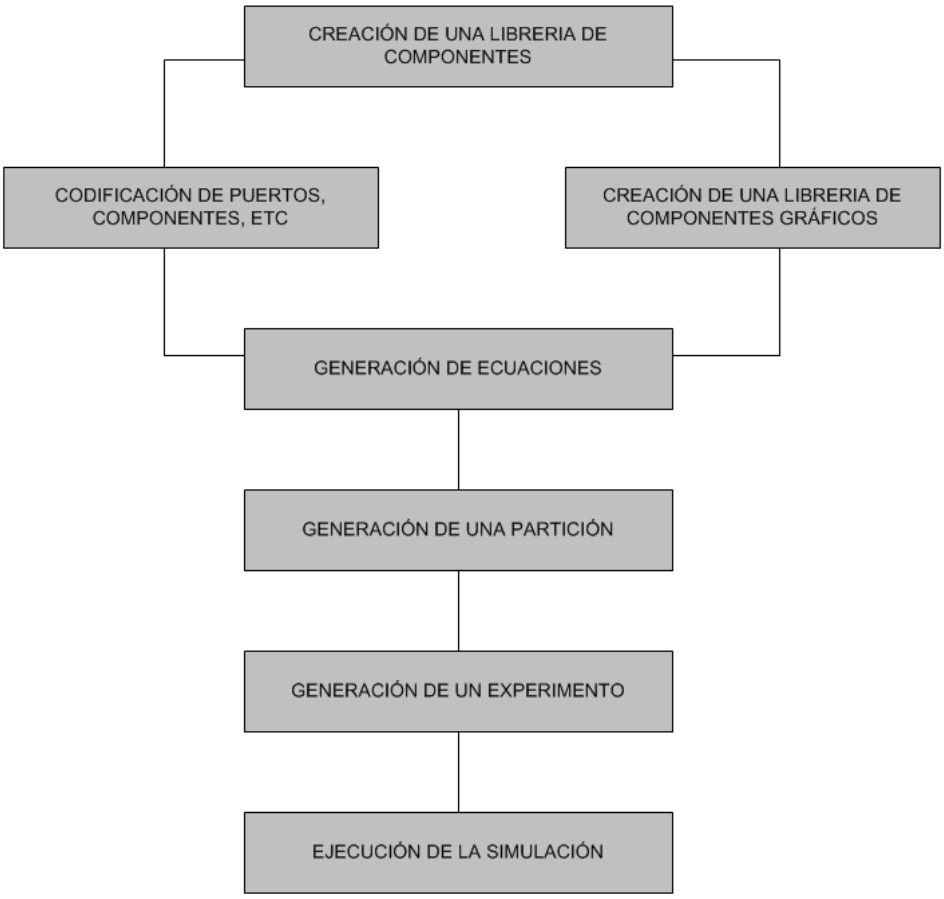


Fig. 2.7.: Forma de trabajar en MOO.

2.2.6.- Ventajas del MOO.

Analizando las características principales que describen la metodología MOO, se pueden enumerar como ventajas las siguientes:

- Flexibilidad. Si se parte del hecho que mediante la definición de clases se establecen módulos independientes a partir de los cuales se pueden definir nuevas clases, entonces se puede pensar en estos módulos como bloques con los cuales es posible construir diferentes programas.

- Reusabilidad. Una vez que se ha definido a la entidad persona para utilizarla en una aplicación no es deseable volver a escribir la definición para la misma entidad persona. Por medio de la reusabilidad se puede utilizar una clase definida previamente en las aplicaciones que sea conveniente. Es claro que la flexibilidad con la que se definió la clase va a ser fundamental para su reutilización.

- Mantenibilidad. Las clases que conforman una aplicación, vistas como módulos independientes entre sí, son fáciles de mantener sin afectar a los demás componentes de la aplicación.

- Extensibilidad. Gracias a la modularidad y a la herencia, una aplicación diseñada bajo el paradigma de la orientación a objetos puede ser fácilmente extensible para cubrir necesidades de crecimiento de la aplicación.

2.2.7.- Limitaciones del MOO.

A pesar de que las ventajas de la programación orientada a objetos superan a las limitaciones de la misma, se pueden encontrar algunas características no deseables en ésta:

- Limitaciones para el programador. Aun siendo la tecnología orientada a objetos antigua, un gran porcentaje de programadores no están familiarizados con los conceptos de dicha tecnología. En otras palabras, la lógica de la programación estructurada sigue siendo predominante en la mayoría de los desarrolladores de software, después de haber revisado de forma breve los principios de la programación orientada a objetos es claro que en ésta se requiere una lógica de pensamiento totalmente diferente a la lógica comúnmente utilizada para la programación estructurada.

- Tamaño excesivo en las aplicaciones resultantes. La gran mayoría de los equipos de cómputo cuentan con capacidades tanto de almacenamiento como de memoria lo suficientemente buena como para ejecutar la mayoría de las aplicaciones que puedan desarrollarse con la tecnología orientada a objetos, sin embargo, existen casos en los que lo anterior no se cumple. Una de las desventajas de la programación orientada a objetos es que cuando se heredan clases a partir de clases existentes se heredan de forma implícita todos los miembros de dicha clase aun cuando no todos se necesiten, lo que produce aplicaciones muy grandes que no siempre encajan en los sistemas con los que se disponga.

- Velocidad de ejecución. Esto tiene que ver con el punto anterior, una aplicación innecesariamente pesada en muchas ocasiones es más lenta de ejecutar que una aplicación conformada únicamente por los módulos necesarios.

CAPÍTULO 3

3. ECOSIMPRO

En la actualidad son muchos los programas usados como herramientas de simulación. El programa que a continuación se presenta y en el que está basado el presente proyecto fin de carrera se denomina EcosimPro. Una vez explicados los conceptos fundamentales sobre simulación y sobre Modelado Orientado a Objetos, se va a tratar de dar una idea de cómo funciona EcosimPro. Para ello se van a explicar sus características principales y las particularidades de esta herramienta de simulación. Esto va a ayudar a comprender cómo se ha creado la librería mecánica rotacional que más adelante se expondrá. También se va a describir cuál es el proceso que sigue EcosimPro para simular un sistema, desde la creación de componentes individuales hasta la simulación de sistemas mediante experimentos.

3.1.- Introducción.

Ecosim comenzó a desarrollarse en 1989 producto de un contrato de Empresarios Agrupados Internacional (EAI) con la *European Space Agency* (ESA) para la simulación de sistemas de control ambiental y soporte de vida en la Estación Espacial Internacional. En 1993 se completó la primera versión de Ecosim y en 1996 la segunda (ambas bajo UNIX). Con el desarrollo de la versión 3.0 (bajo Windows) se cambió el nombre de la herramienta de Ecosim a EcosimPro. La primera versión comercial apareció a finales de 1999. Hoy en día se puede encontrar en el mercado la versión 4.8. Esta es la herramienta de modelado estándar de la ESA en las áreas de *Environmental Control and Life Support System* (ECLSS), Propulsión (Satélite y Cohete) y Sistemas Biológicos para misiones de larga duración [VAZ10]. Para la realización del presente proyecto fin de carrera se ha utilizado la versión 4.6. de EcosimPro.

EcosimPro es una herramienta de modelado y simulación multidisciplinar de última generación, con un lenguaje de modelado (EL, EcosimPro Langauge) orientado a objetos que genera código C++ siguiendo un estándar desarrollado por la ESA. La descripción del modelo puede hacerse usando EL o usando el editor gráfico. Este dispone de un interfaz hombre-máquina que facilita la labor de creación de modelos de una manera intuitiva.

En este programa, el lenguaje de modelado EL impone el encapsulamiento de la información. Como ya se ha comentado en el Capítulo 2, los únicos elementos visibles de un componente en la fase de modelado son los de su interfaz pública, es decir, sus

puertos, sus parámetros y sus datos. EL también permite la composición de componentes y la especialización mediante herencia múltiple.

3.2.- Comunicación con entornos externos.

El uso de esta herramienta de simulación puede resultar más o menos complejo dependiendo del fin buscado, se van a clasificar los niveles de usuario en base al grado de dificultad de uso. En un primer nivel aparecen modeladores de librerías, personas que requieren conocer a fondo la matemática de los componentes y el lenguaje de modelado; en un segundo nivel se encuentran los usuarios de librerías ya terminadas, lo que hacen es diseñar sistemas gráficamente; en el tercer nivel estarían las personas que crean múltiples experimentos sobre un modelo matemático ya cerrado; y el cuarto y último nivel es para la gente que usa modelos de EcosimPro desde otros entornos externos, ya que existe la posibilidad de trabajar con multitud de programas informáticos relacionados con esta herramienta de simulación como son Microsoft Excel, Matlab, clases C++, etc. En la Figura 3.1. aparece un gráfico donde se ven los diferentes programas que soportan la funcionalidad de EcosimPro.

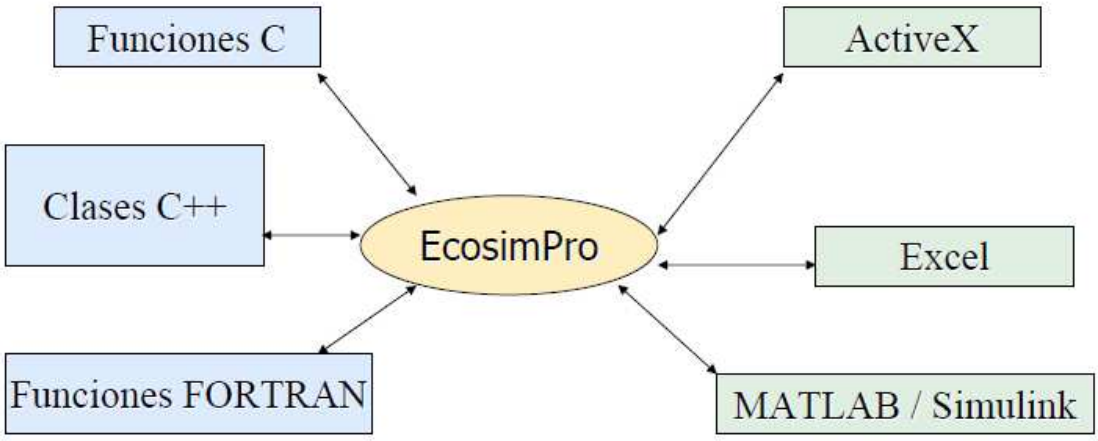


Fig. 3.1.: Interfaz con programas externos.

Gracias a estas conexiones es posible usar toda la potencia que proporciona la herramienta sin necesidad de emplear el entorno de simulación y puede incluirse su funcionalidad en aplicaciones independientes desarrolladas por el usuario. En la Figura 3.2. hay un ejemplo de la herramienta EcoExcel, que se usa para exportar modelos de EcosimPro a Microsoft Excel a través de un "*add-in*" creado para conectarse a modelos generados desde la herramienta.

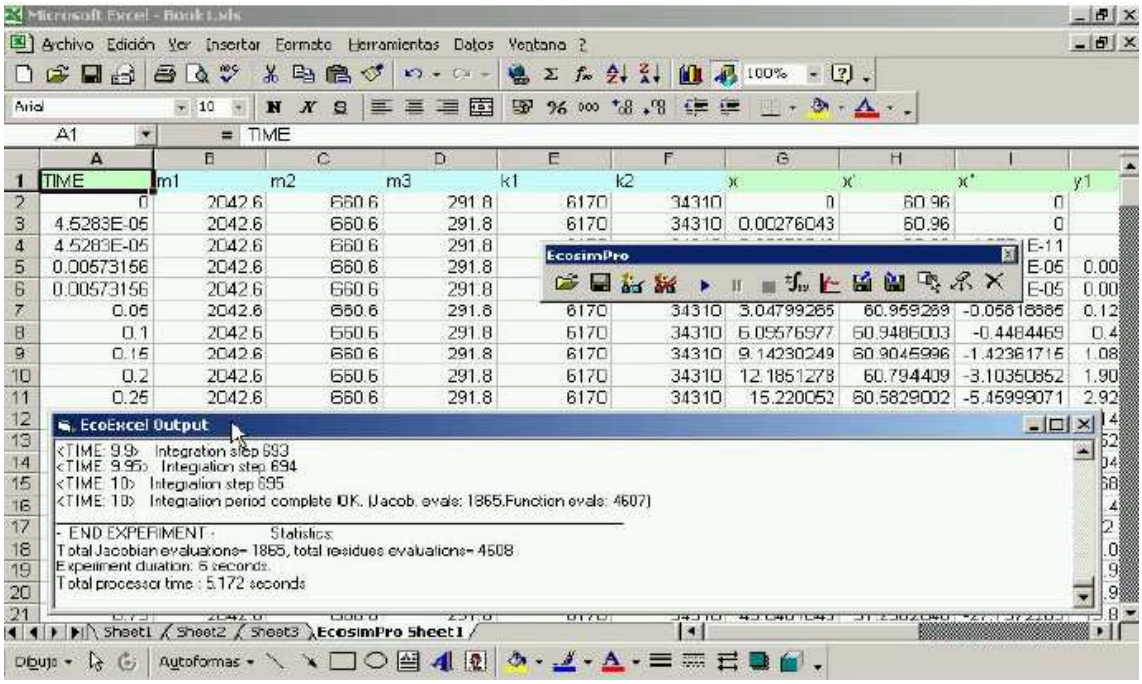


Fig. 3.2.: "Add-in" de Excel generado desde EcosimPro.

3.3.- Modelado de componentes y sistemas.

Un "*workspace*" es un espacio de trabajo donde se pueden agrupar los modelos, librerías, simulaciones, etc. Una librería es un conjunto de componentes agrupados por su temática, por afinidad, o por capricho simplemente. Todo componente debe estar incluido en alguna librería. A la hora de crear componentes, EcosimPro da la posibilidad de elegir entre sus tres modos de vista: "*Code View*", donde se permite introducir, leer, modificar, etc., el código de los componentes elaborados; "*Schematic View*", en la que se permite crear componentes por agregación de otros más básicos ya creados utilizando un entorno gráfico; y "*Simulation View*", es la vista de simulación donde se crean los experimentos y a la que se accede cuando se tienen creados los componentes.

Antes de empezar a crear un componente, se deben crear los puertos pertenecientes a ese componente de forma independiente, de tal manera que esos puertos puedan usarse para otros componentes que compartan las mismas entradas o salidas. Un componente tiene una interfaz pública, que será la única información que comparta con otros componentes, esta interfaz pública se define por los puertos, que son un conjunto de variables que pueden compartirse con otros componentes. El resto de elementos del componente: variables locales, ecuaciones, etc., permanecen ocultas para los demás componentes. Por ejemplo, en un modelo de un sistema mecánico rotacional, se puede definir un PORT que incluya las variables de ángulo de rotación y de momento torsor. Un muelle rotacional o un embrague, por ejemplo, tendrían dos PORTS de este tipo. Si se conectan dos de estos componentes por sus puertos se puede conseguir que los ángulos de giro y los momentos de los extremos conectados queden relacionados de forma adecuada mediante ecuaciones generadas automáticamente por EcosimPro que se

incorporan al modelo. La definición de un PORT se suele introducir en la librería para que pueda ser utilizado por los componentes incluidos en ella.

La definición de los puertos incluye la declaración de sus variables, donde se especifica el tipo (REAL, BOOLEAN, etc.), la forma en que debe ser conectada (EQUAL, SUM) y el sentido del puerto (IN, OUT). Al definir el puerto también se puede especificar cierto tipo de restricciones acerca de la forma en que puede ser conectado. Como ejemplo, al conectar dos elementos mecánicos rotacionales se generará una ecuación que indique que los ángulos de giro de los dos PORTS son iguales y otra que indique que la suma de los momentos es cero. La sintaxis de la definición del PORT mecánico rotacional es la de la Figura 3.3.

```
PORT mech_rot "1D rotational flange"
   SUM   REAL tau UNITS "N*m"    "Torque (N*m)"
   EQUAL REAL phi UNITS "rad"    "Absolute angular position (rad)"

END PORT
```

Fig. 3.3.: Modelo de un puerto mecánico rotacional.

El puerto PORT incluye dos variables reales, ángulo de giro *phi* y momento torsor *tau*. Como puede observarse, al final de cada variable aparece entre comillas las unidades y la descripción. Las “palabras reservadas” EQUAL, que indica que las variables con este identificador de todos los puertos conectados tendrán el mismo valor, y SUM, que indica que la suma de todas las variables de este tipo de los puertos conectados sumarán cero, indicarán las ecuaciones de conexión que se crearán automáticamente al conectar los puertos de dos o más componentes y que se encargará de relacionar las variables de los dos puertos conectados. Por ejemplo, si se conectan dos PORTS del tipo *mech_rot* llamados C1 y C2 las ecuaciones generadas serán las Ecuaciones (3.1.) y (3.2.):

$$C1.phi = C2.phi \qquad (3.1.)$$

$$C1.tau + C2.tau = 0 \qquad (3.2.)$$

En EcosimPro hay que diferenciar entre dos tipos de componentes al declararlos: los abstractos (ABSTRACT COMPONENT) y los concretos (COMPONENT). Los componentes abstractos son aquellos que por sí mismo no representan ningún componente real, más adelante se mostrarán varios ejemplos, y sólo pueden ser usados para crear otros componentes. Los componentes concretos son aquellos que representan por sí mismo una realidad física.

En la declaración de los componentes se organiza en diferentes secciones la definición de los parámetros y de las constantes (sección DATA), la declaración de las variables (sección DECLS) y de los puertos (sección PORTS), la parte continua (sección CONTINUOUS), la discreta (sección DISCRETE) y las sentencias de inicialización a ejecutar secuencialmente antes del comienzo de la simulación (sección INIT). Un componente es el elemento fundamental en EcosimPro, en él se define el comportamiento continuo y discreto de un modelo. Puede formar parte de otro

componente más complejo mediante interconexión a través de sus puertos. Puede haber varios bloques opcionales dentro de un componente dependiendo de su comportamiento:

- **IS_A:** permite definir la herencia. La forma de describirlo es la siguiente:

  COMPONENT C2 IS_A C1.

  esto quiere decir que el componente C2 hereda del componente C1.

- **PORTS:** define las variables accesibles desde el exterior del componente. Va con identificadores IN/OUT que sirven para definir el sentido de las variables de flujo (el signo).

- **DATA:** Constantes y parámetros conocidos del componente. Este valor puede ser modificado más adelante en la simulación.

- **DECLS:** Variables declarativas locales que se usan principalmente en la zona de comportamiento discreto o continuo. Estas variables, o bien son variables dinámicas que el resolvedor de ecuaciones de EcosimPro deberá calcular, o bien son variables de contorno que el usuario deberá especificar en el experimento.

- **INIT:** Secuencia de instrucciones que hay que ejecutar antes de comenzar la simulación.

- **DISCRETE:** Define el comportamiento discreto de un componente.

- **CONTINUOUS:** Con este bloque, que se puede introducir en un PORT o en un COMPONENT, se permite dar valor a alguna variable en función de otras. Aquí se añaden todas las ecuaciones que implican variables que cambian de forma continua a lo largo del tiempo.

Seguidamente, en la Figura 3.4. se puede ver un ejemplo con el código de un componente abstracto.

```
ABSTRACT COMPONENT R_Two_Ports
"Abstract class for definition of components with two rotational ports"
   PORTS
      IN  mech_rot m_in     "Left / driving mech_rot"
      OUT mech_rot m_out    "Right / driven mech_rot"

END COMPONENT
```

Fig. 3.4.: Modelo de un componente abstracto genérico.

El componente anterior por sí solo no dice nada, se puede ver por su definición entre comillas, que es un componente abstracto con dos puertos mecánicos rotacionales, uno de entrada y otro de salida. Este componente abstracto puede ser usado dentro de otros componentes para que este adquiera sentido. En la Figura 3.5. se muestra un componente que hereda de un componente abstracto.

```
COMPONENT R_Inertia IS_A R_Rigid
"Rotating mass with inertia"
  PORTS
     OUT mech_rot m_out
  DATA
     REAL I            UNITS "kg*m^2"
     REAL phi_0 = 0    UNITS "rad"

  DECLS
    REAL alpha         UNITS "rad/s^2"

  INIT
     phi = phi_0

  CONTINUOUS

     phi'' = alpha
     I * alpha = m in.tau - m out.tau
   DISCRETE
   WHEN TIME == 5 THEN
        I = 500000

   END WHEN

   WHEN TIME == 5.5 THEN
        I = 0
   END WHEN

END COMPONENT
```

Fig. 3.5.: Modelo de un componente concreto genérico.

Se observa que dentro de este modelo de componente genérico, que representa una masa rotacional con inercia, se han utilizado todos los bloques que han sido explicados con anterioridad. Una vez definidos los componentes en el lenguaje de programación hay que compilarlos, esto significa chequear la sintaxis de sus modelos y es algo que EcosimPro hace automáticamente. Si el código es correcto y no ha habido errores se dará el visto bueno para seguir con la simulación, de lo contrario, habrá que revisar el modelo para modificarlo y hacer que funcione correctamente.

En EL se definen tres tipos diferentes de sentencias: secuenciales, continuas y discretas. Las sentencias secuenciales son ejecutadas sin modificar su orden. Se emplean en las inicializaciones, en funciones y en las acciones de eventos discretos, en aquellos casos en que se requiere un orden estricto de ejecución. Las sentencias secuenciales son las habituales en los lenguajes de programación: asignaciones, llamadas a funciones, IF-THEN-ELSE, WHILE, FOR, etc. Las sentencias continuas se emplean para expresar las ecuaciones algebraico-diferenciales de forma acausal y son ordenadas por los algoritmos internos de EcosimPro. Se permite la descripción de ecuaciones escalares, vectoriales y matriciales. Aparte, soporta la descripción de determinados tipos de sistemas de estructura variable. Las sentencias discretas se usan para expresar eventos. EL permite realizar en tiempo de simulación chequeos de integridad, de modo que es posible incluir, en la parte secuencial o discreta, sentencias que generen avisos o mensajes de error cuando deja de satisfacerse determinada condición.

Ya compilado el modelo es necesario crear una partición, EcosimPro también hace esto de forma automática. Crear una partición consiste en ordenar las ecuaciones para poder crear el experimento. Esto genera un modelo matemático por defecto para el componente que hay que validar. Con esto se puede ver que la partición es correcta. La

forma de validarlo es igual de simple para el usuario que compilar el componente o crear una partición, es decir, automáticamente.

Antes de crear el experimento es necesario crear un componente que incluya la definición de los elementos del sistema y las conexiones de los puertos de los elementos definidos. Para crear estos componentes la sintaxis a utilizar es la siguiente:

- **USE:** hay que hacer mención a la, o las, librerías donde se encuentran los modelos de los elementos que van a formar el sistema.

- **COMPONENT:** aquí se debe incluir el nombre del componente a partir del cual se generan los experimentos.

- **TOPOLOGY:** en este apartado se introducen todos los componentes que van a formar el sistema. Hay que darles un nombre para luego poder hacer referencia a ellos. Aquí también se podrá crear o modificar el valor de las constantes de los componentes, escribiendo el valor siempre entre paréntesis.

- **CONNECT_TO:** aquí hay que escribir todas las conexiones entre los componentes del sistema. Hay que unir uno o varios puertos de uno o varios componentes con uno o varios puertos de uno o varios componentes para formar el sistema a estudiar. La forma de unir dos componente mediante un puerto es poniendo primero el nombre del componente seguido de un punto y el puerto de salida con el nombre del otro componente seguido de un punto y el puerto de entrada.

Un ejemplo del modelo de un sistema de componentes en el lenguaje de EcosimPro sería el mostrado en la Figura 3.6.

```
USE ROTACIONAL

COMPONENT SISTEMA_ROT_2GDL
   TOPOLOGY
      PARED_ROT P1 (phi_0=0)
      MUELLE_ROT M1 (k=157913.4, phi_rel0=0)
      INERCIA_ROT_2PUERTOS I2 (I=4000, phi_0=0 ,omega_0=0)
      DAMPER_ROT D1 (b=12566.36, phi_rel0d=0)
      MOMENTO_ROT PAR1
      MUELLE_ROT M2 (k=157913.4, phi_rel0=0)
      INERCIA_ROT_2PUERTOS I1 (I=4000, phi_0=0 ,omega_0=0)

      CONNECT P1.p TO M2.q
      CONNECT M2.p TO I1.p
      CONNECT I1.q TO M1.q
      CONNECT I1.q TO D1.q
      CONNECT M1.p TO I2.p
      CONNECT D1.p TO I2.p
      CONNECT I2.q TO PAR1.p
END COMPONENT
```

Fig. 3.6.: Modelo de un sistema de componentes.

En la Figura 3.6. se ha representado un sistema rotacional de dos grados de libertad formado por una pared, dos muelles, dos discos de inercia y un amortiguador rotacional.

Finalizada la creación del componente que define el sistema se puede crear un experimento para la partición generada por defecto. En la imagen de la Figura 3.7. se

puede ver, en la pantalla *"Code View"* de EcosimPro, como se modela un componente básico, en este caso un péndulo oscilante, dentro de la librería DEFAULT_LIB.

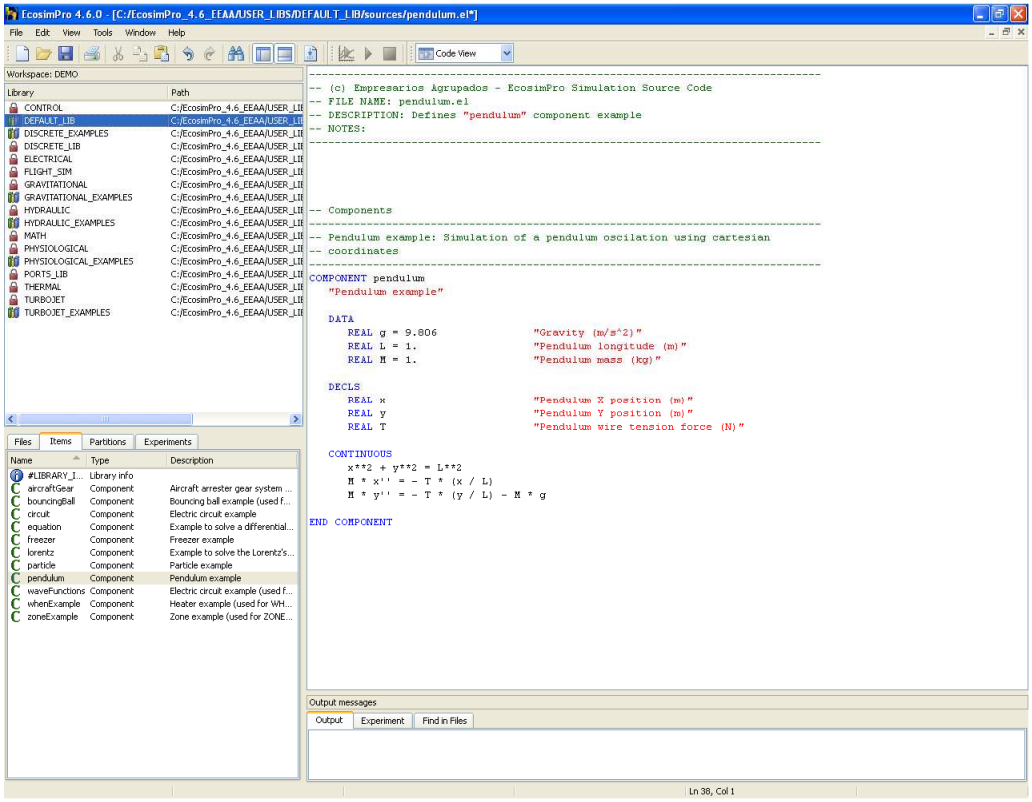


Fig. 3.7.: "Code View" para modelar componentes básicos.

Esta es la forma de crear modelos de sistemas usando el lenguaje de programación EL. Ya se ha comentado que existe otro método más interactivo de crear estos modelos en EL mediante la pantalla *"Schematic view"*, el entorno de modelado gráfico de EcosimPro. El siguiente bloque trata sobre este tipo de modelado que pertenece a otro nivel de usuarios.

3.4.- Representación gráfica de componentes y sistemas.

Una de las facilidades que ofrece EcosimPro a los usuarios es la posibilidad de crear sistemas a partir de elementos gráficos que han sido diseñados previamente. Las librerías se crean como se ha explicado hasta ahora, es decir, programando códigos para los puertos, para las clases abstractas y para los componentes. Después de haber creado las librerías deseadas es posible asignar a cada componente de estas librerías un icono gráfico de manera que, uniendo varios componentes por sus conexiones (puertos), se puedan obtener componentes más complejos o incluso sistemas de componente. Esto también se puede hacer mediante programación de códigos, pero resulta mucho más sencillo y rápido hacerlo mediante el entorno gráfico que facilita EcosimPro. Este entorno gráfico de que se va a hablar a continuación se llama EcoDiagram. Cuando se crea un componente complejo o un sistema de componentes con la herramienta gráfica EcoDiagram, el compilador del propio programa es el encargado de traducir los

símbolos gráficos y sus conexiones al código adecuado, evitándole todo este proceso al usuario.

Para asociar símbolos gráficos a componentes con EcoDiagram lo primero que hay que hacer es seleccionar la librería dentro de la cual se quiere trabajar. Allí van a estar los componentes a los que se pretende asignar iconos. El presente proyecto fin de carrera se ha centrado en elementos mecánicos rotacionales, con lo cual, si lo que se quiere es crear iconos a sus componentes se deberá abrir la librería mecánica rotacional: *Rotational Library*, creada previamente. Como se ve en la Figura 3.8., con seleccionar esta librería en el "*Workspace*" es suficiente.

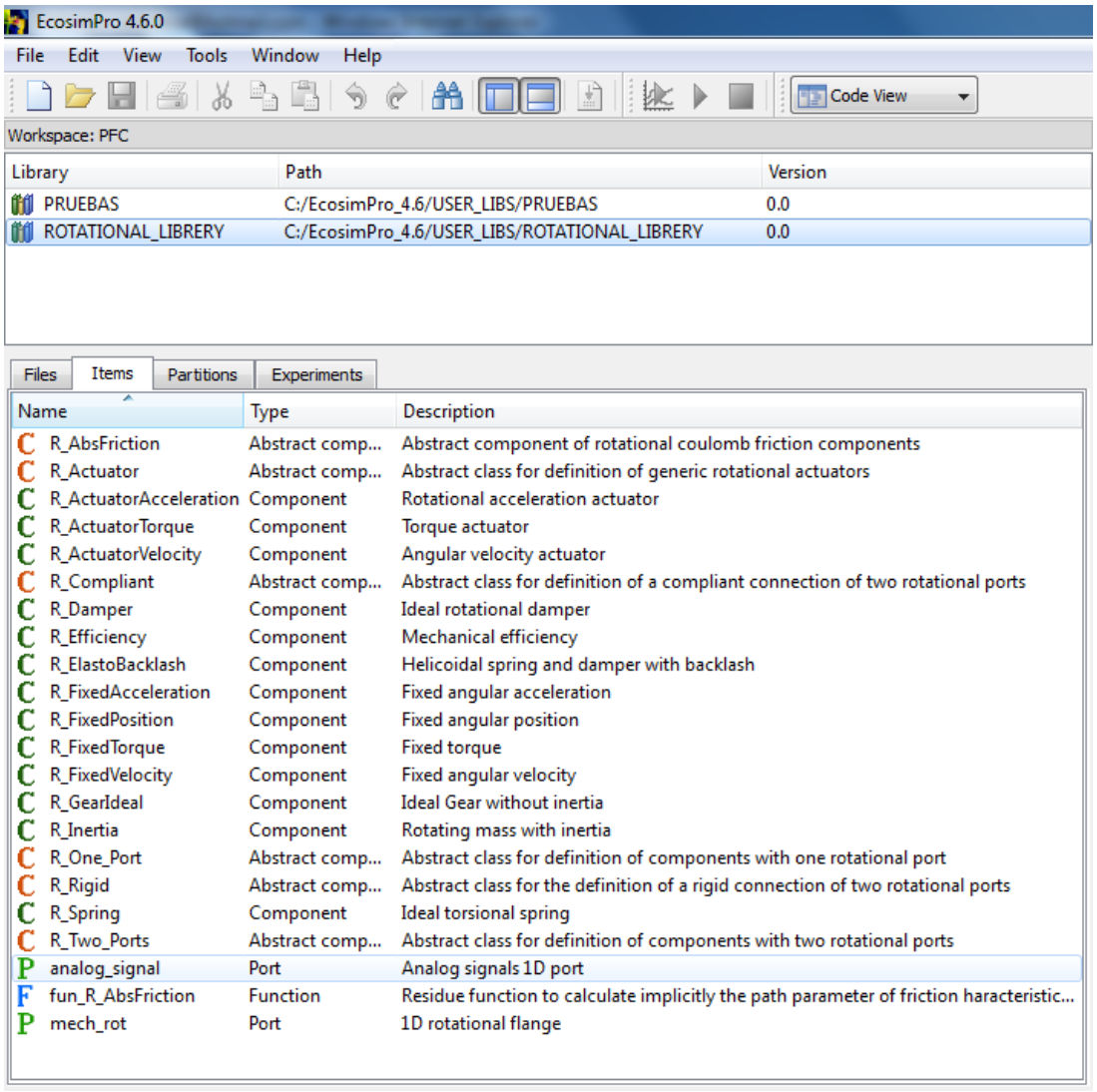


Fig. 3.8.: Ventana de selección de librerías.

Con la ayuda de los comandos que ofrece el programa se pueden crear los símbolos gráficos que se desee. Aquí se puede emplear más o menos tiempo dependiendo de los gustos del usuario, pero es conveniente ir siempre componente por componente, empezando por lo más sencillo y acabando con lo más complicado. Primero se crearan los puertos, después los componentes sencillos y para acabar los componentes más complejos.

Para asignar iconos a los diferentes componentes hay que seleccionar el botón mostrado en la Figura 3.9. en la pantalla *"Schematic view"*.

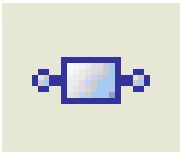


Fig. 3.9.: Botón para asignar iconos a componentes.

Si lo que se pretende es crear los iconos de los puertos, hay que seleccionar simplemente un color que lo identifique, esto ayudará a diferenciar entre un puerto mecánico, un puerto analógico, un puerto eléctrico, etc. En la Figura 3.10. se aprecia un puerto eléctrico de color rojo, de un simple vistazo se puede ver qué tipo de puertos forman el componente.

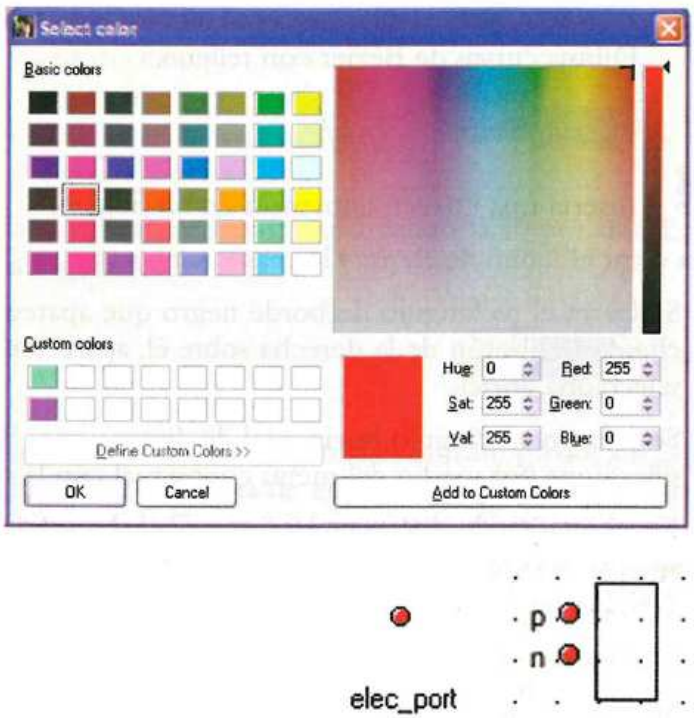


Fig. 3.10.: Ventana de selección de color, icono y lienzo para puertos.

Una vez creados los iconos de los puertos se crean los iconos para los componentes. Esto es un poco más complejo, pero con la experiencia necesaria y con las numerosas ayudas que ofrece EcosimPro se convierte en una tarea mucho más sencilla. Para crear un icono de un componente hay que seleccionar el componente, después hay que ir al lienzo y dibujar algo representativo del componente en cuestión, algo que sea intuitivo a simple vista. En cuanto se selecciona un componente, EcosimPro abre el lienzo directamente y ofrece una barra de herramientas para la construcción del icono gráfico. En dicha barra de herramientas hay diferentes comandos que ayudan a crear los iconos, estos son los mostrados en la Figura 3.11.

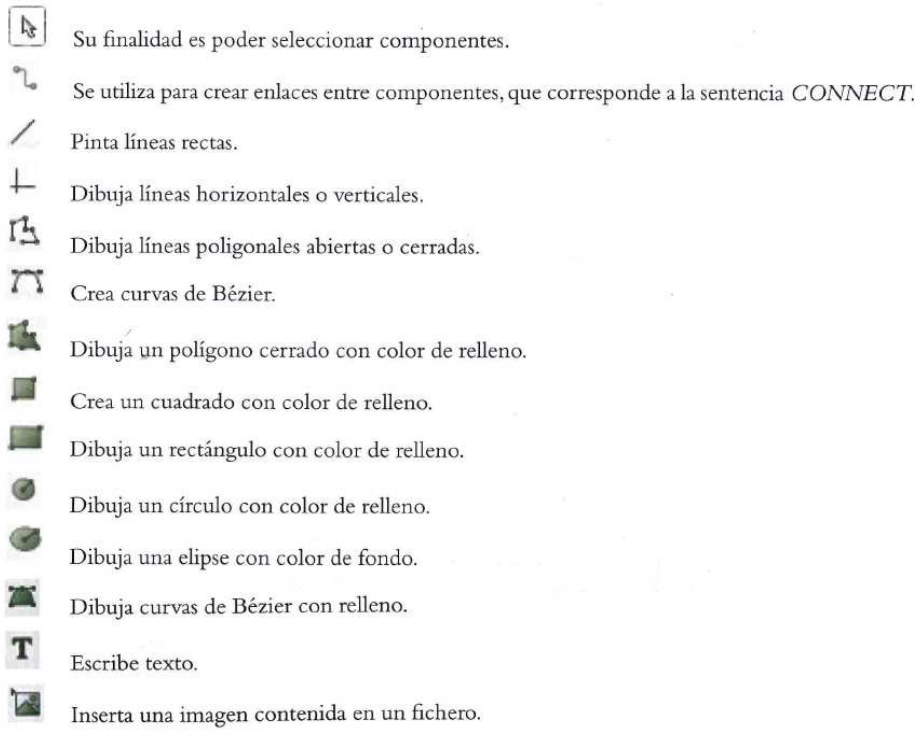


Fig. 3.11.: Comandos para crear iconos.

Como se ve en el último comando, es posible asociar a un componente una imagen desde un archivo externo en vez de dibujar en el lienzo un icono. Esto puede ser mucho más cómodo para el usuario, ya que con una foto del componente puede tener su representación gráfica. En la Figura 3.12. se expone una imagen de un lienzo en la cual se puede ver la representación gráfica de una resistencia eléctrica con dos puertos eléctricos y los comandos de creación previamente citadas.

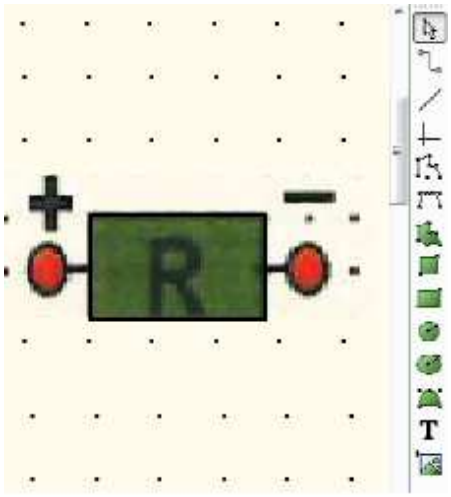


Fig. 3.12.: Lienzo y componente creado.

Una vez se han asignados iconos a los componentes se va a mostrar cómo es posible construir sistemas de forma gráfica a partir de los iconos de los componentes creados previamente y de las uniones entre ellos. Después, como ya se ha dicho, el compilador del propio programa será el encargado de traducir los símbolos gráficos y sus conexiones al código adecuado.

Al igual que a la hora de asociar iconos a componentes hay que seleccionar la librería con la cual se quiere trabajar, a la hora de crear sistemas hay que seleccionar el entorno gráfico en el que se va a desarrollar el sistema. Esto se hace eligiendo la vista esquemática, es decir, "*Schematic View*". Una vez abierta esta vista hay que elegir el icono del puerto o del componente que se quiere usar y llevarlo al lienzo. Una vez allí, para conectarlo con otros iconos de otros puertos o componentes y formar un sistema, hay que ir uniéndolos mediante líneas, como se ve en la Figura 3.13.

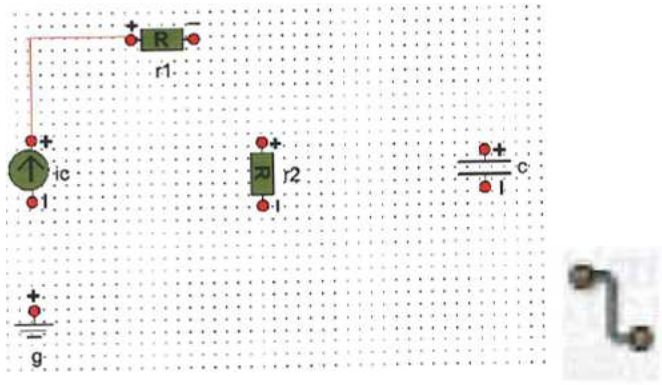


Fig. 3.13.: Unión de componentes.

Una vez dibujados todos los componentes y conectados todos sus puertos mediante líneas, ver Figura 3.14., el sistema está creado.

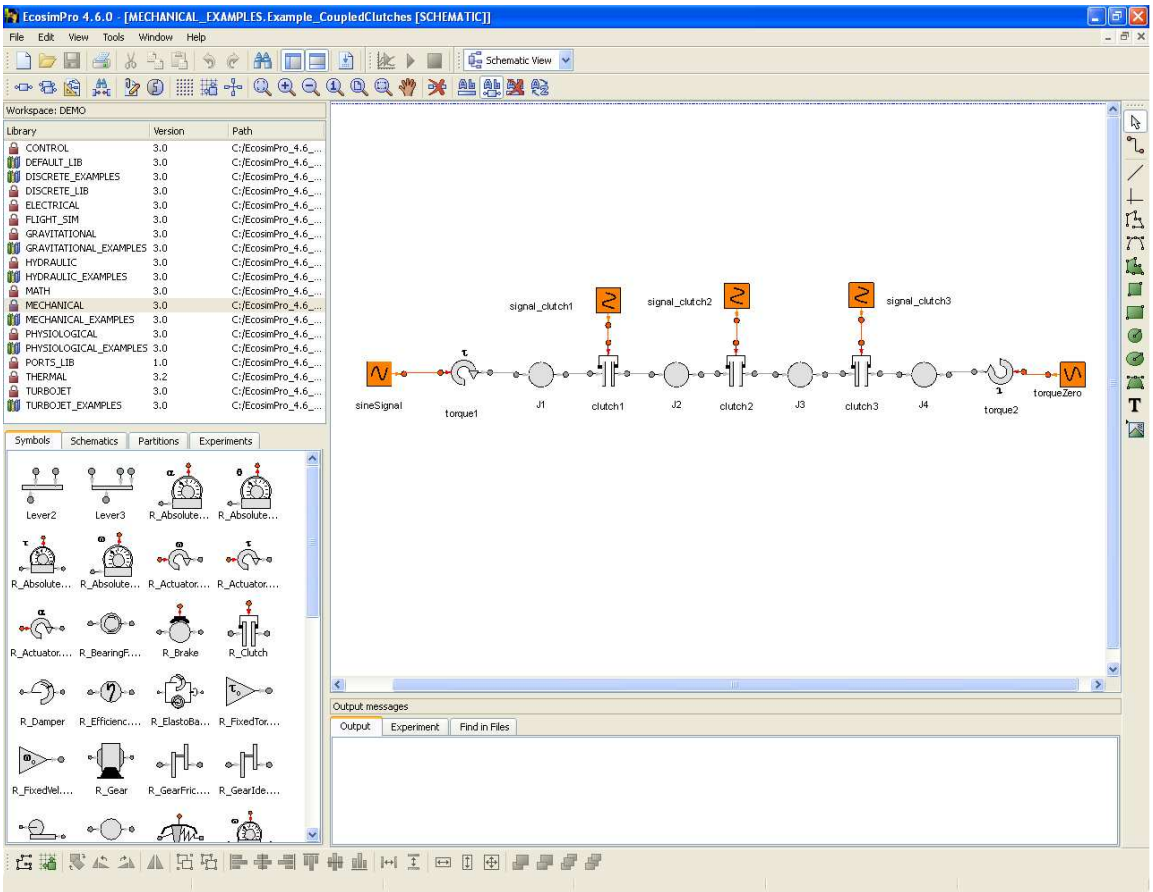


Fig. 3.14.: "Schematic View" para modelar sistemas en el entorno gráfico.

Lo último que habría que hacer para realizar los diferentes experimentos sería dar valores a las constantes de cada uno de los componentes que forman el sistema pinchando dos veces sobre ellos y añadiendo el valor en tablas como en la mostrada en la Figura 3.15.

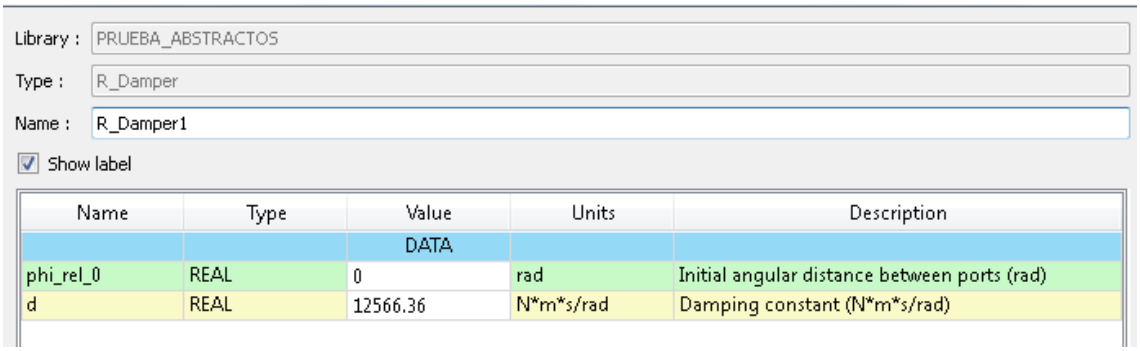


Fig. 3.15.: "Schematic View" para modelar sistemas en el entorno gráfico.

3.5.- Simulación de sistemas.

Con estas operaciones termina la fase de modelado y se pasa a la operación de simulación. EcosimPro genera por sí solo diferentes experimentos, los cuales se pueden variar al gusto del usuario. EcosimPro dispone de un lenguaje para la descripción de experimentos que permite especificar el estado inicial del sistema y las condiciones de contorno que representan el estado del entorno, integrar el modelo entre un instante inicial y uno final, calcular el estado estacionario a partir de las condiciones en un determinado instante, optimizar parámetros, especificar el tipo de informe que se desea obtener como resultado de la simulación, etc.

La sintaxis de un experimento consta de varios bloques claramente diferenciados y que se podrán ver en el ejemplo expuesto más adelante:

- **INIT:** es la parte donde se inicializan las variables que durante el proceso de creación de la partición EcosimPro ha detectado que son variables de estado algebraicas y, por lo tanto, tienen que ser inicializadas.

- **BOUNDS:** es el lugar donde hay que dar valores a las variables que EcosimPro detecta como variables. Estas no pueden despejarse de ninguna ecuación, o no están definidas en ningún sitio o no son variables de estado. Son las llamadas variables de contorno o de frontera.

- **BODY:** es el cuerpo de la simulación, donde se da valores tales como la duración de la simulación.

En la Figura 3.16. se muestra el código de un experimento que ha sido generado directamente por EcosimPro, el cual no es único y puede variar en función de lo que se quiera conocer. Para modificarlo se podrían inicializar las variables de otra manera, se

podría cambiar el tiempo de finalización del experimento, se podrían añadir otras ecuaciones adicionales a las que ya existen, etc.

```
EXPERIMENT exp1 ON pipeCircuit.math1
   DECLS
   INIT
   BOUNDS
      -- Set equations for boundaries: boundVar
      Pipe1.hp_in.p = 101325
      Pipe1.hp_in.w = 10 * (1 + sin(TIME))
      Pipe5.hp_out.w = 2 * (1 + sin(TIME))
      Pipe6.hp_out.w = 3 * (1 + sin(TIME))
      Pipe7.hp_out.w = 4 * (1 + sin(TIME))

   BODY
      -- report results in file reportAll.rpt
      REPORT_TABLE("reportAll.rpt", "*")
      -- for example integrate the model 15 seconds
      TIME = 0
      TSTOP = 15
      CINT = 0.1
      INTEG()
END EXPERIMENT
```

Fig. 3.16.: Ejemplo de experimento en EcosimPro.

EcosimPro genera automáticamente las ecuaciones del modelo completo, detectando automáticamente las variables equivalentes y las ecuaciones triviales. Detecta los problemas que puedan surgir de índice superior y si estos problemas son debido a ligaduras lineales entre las variables que aparecen derivadas, lo resuelve eliminando del modelo estas ligaduras y las variables que aparecen derivadas. En caso contrario, aplica el algoritmo de Pantelides, que es un algoritmo simbólico que reduce el índice de perturbación y muestra qué variables pueden ser seleccionadas como variables de estado, indicando cuantas deben escogerse y propone una selección [PAN88].

Una vez que el problema no tiene índice superior, el programa compara el número de ecuaciones y de incógnitas del sistema. Si el número de ecuaciones es mayor que el de incógnitas muestra un mensaje de error indicando el conjunto de ecuaciones redundantes. Si hay más incógnitas que ecuaciones es preciso especificar condiciones de contorno adicionales. EcosimPro muestra una posible elección de variables sobre las que imponer las condiciones de contorno. Gracias a todas estas ayudas, se facilita al usuario la simulación de los experimentos deseados.

El *Graphical User Interface* (GUI) de EcosimPro proporciona un monitor del experimento, "*Experiment Monitor*", que permite conocer y cambiar el valor de las variables durante la simulación. En la Figura 3.17. se muestra un monitor real con diferentes tipos de experimentos.

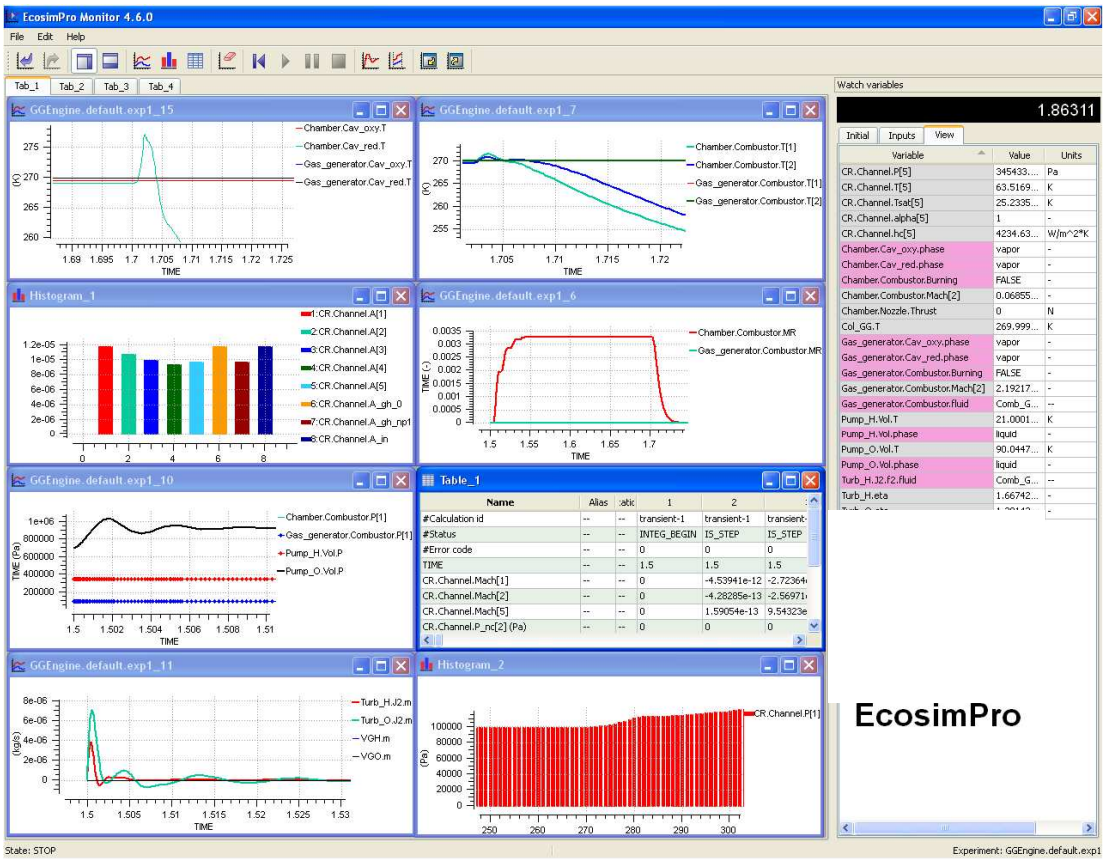


Fig. 3.17.: Monitor para realizar diferentes simulaciones.

3.6.- Comparación entre lenguajes de programación: EcosimPro Language (EL) y Modelica.

Una vez visto cómo funciona EcosimPro se va a comparar el lenguaje que usa este, EcosimPro Language (EL), con otro lenguaje muy usado en modelado y simulación de sistemas, Modelica. Es importante comparar estos dos lenguajes orientados a objetos y basados en ecuaciones para el modelado de sistemas físicos continuos y discretos porque para la creación de las librerías en EcosimPro se han utilizado como ayuda muchos códigos de puertos, componentes y demás elementos de Modelica.

Teniendo en cuenta las propiedades más importantes de los lenguajes orientados a objetos, se va a presentar un estudio a nivel general para dar a conocer cuáles son las diferentes características que cada lenguaje incorpora. Esta parte concluirá con un apartado en el cual se habla sobre las características que podrían ser incorporadas o modificadas en EL, con el fin de mejorar las prestaciones de este lenguaje de modelado y simulación de sistemas.

Modelica es un moderno lenguaje de modelado orientado a objetos que permite la modelación de sistemas físicos complejos tales como sistemas mecánicos, eléctricos, hidráulicos, térmicos o de control. Es un lenguaje de código abierto desarrollado por *The Modelica Association* [MOD02], que también desarrolló la *Modelica Standard*

Library, y que contiene más de 780 modelos de componentes genéricos y 550 funciones multipropósito (referencia a la versión 3.0 de febrero de 2008). A pesar de que Modelica es similar a lenguajes de programación orientados a objetos, como *C++* y *Java*, difiere con ellos en dos importantes aspectos. Primero, Modelica es un lenguaje de modelación y no un verdadero lenguaje de programación. Y segundo, las clases escritas en Modelica no son compiladas en el sentido usual de la palabra, son traducidas en objetos que pueden ser utilizados en sistemas de simulación y que no son específicos del lenguaje, aunque ciertas capacidades necesarias sean delineadas por el lenguaje.

Modelica se creó por la necesidad de definir un lenguaje de modelado y simulación estándar. Posteriormente se creó EL para la herramienta de modelado EcosimPro. Ambos lenguajes están capacitados para modelar y simular sistemas dinámicos continuos, discretos o híbridos de manera acausal [FRI06].

Este estudio se centrará en los aspectos diferenciadores entre ambos lenguajes a nivel conceptual obviando detalles que estén relacionados con características que no sean relevantes en las actividades de modelado y simulación.

3.6.1.- Tipos y propiedades de datos.

Los dos lenguajes admiten los siguientes tipos fundamentales de datos: Booleanos (BOOLEAN en EL), Enteros (INTEGER), Reales (REAL) y Cadena de Caracteres (STRING). Sin embargo, la primera diferencia aparece cuando se habla de los tipos de datos fundamentales sobre tablas en 1D, 2D, 3D (TABLE 1D, 2D, 3D) y de punteros a funciones (FUNC PTR). Estos tipos de datos los contempla EL pero no Modelica. Modelica incorpora mecanismos para implementarlos como tipos compuestos y sustituir su funcionalidad.

Para Modelica los tipos básicos de datos se consideran como clases preconstruidas compuestas por una serie de atributos que modelan su comportamiento. Estos atributos están definidos en el lenguaje Modelica para que las herramientas que los implementan puedan utilizarlos, aunque no necesariamente lo hagan (la herramienta utilizada normalmente para leer Modelica es *Dymola*). Sin embargo, EL no trata los datos básicos como clases, sino como datos básicos del lenguaje que formarán parte de otros componentes.

Otra diferencia entre los dos lenguajes aparece al referirse a datos enumerados y datos derivados. Ambos lenguajes contemplan los tipo enumeraciones pero los tipo subconjunto sólo están definidos para EL. Con respecto a los tipo "*arrays*" (colección de variables del mismo tipo) hay algo que EcosimPro debería mejorar, y es la limitación que tiene con los elementos que pueden estar contenidos en una matriz o tabla. Es cierto que en las últimas versiones de EcosimPro se han añadido nuevas mejoras al lenguaje de modelado como capacidad de usar "*arrays*" de componentes, funciones de acceso a dimensiones de "*arrays*" en tiempo de ejecución, ordenación inteligente de "*arrays*" en los "*reports*" (informes), nuevas funciones para escribir resultados en un formato profesional, etc., pero EL todavía restringe algún tipo básico y enumerado como elementos dentro de un "*array*". Modelica, además de estos tipos, permite que las clases y modelos estén dentro de "*arrays*". Esta limitación puede llegar a plantear

dificultades con EL en el modelado de sistemas formados por estructuras regulares dispuestas en "*arrays"* e interconectadas entre sí.

La variabilidad de los distintos tipos de datos está presente en ambos tipos de lenguaje. La única diferencia existente es a nivel sintáctico. Ambos permiten definir variables como constantes, que una vez definidas nunca cambian su valor; como parámetros, que una vez declaradas e inicializadas no cambian su valor durante una simulación; como variables discretas, que cambian su valor solamente en los instantes donde se producen eventos discretos en una simulación y en los intervalos de tiempo entre eventos mantienen su valor; y como variables continuas, que cambian su valor en cualquier instante de la simulación.

3.6.2.- Terminales o puertos.

Entre los terminales en Modelica y los puertos en EL hay una gran diferencia. Como ya se ha comentado, los terminales o puertos permiten comunicar unos componentes con otros dentro de un mismo sistema. En este tipo de componentes, EL muestra una mayor flexibilidad que Modelica. EL permite definir mediante unas palabras reservadas (IN y OUT) la direccionabilidad de las variables de los terminales. EL permite introducir ecuaciones que relacionan las variables y restringir el número de conexiones en un puerto.

En Modelica, la direccionabilidad está definida en el lenguaje por convenio. Tampoco se permite que en los terminales existan ecuaciones, de hecho todos los modelos en Modelica son clases. Los conectores son clases que están restringidas a no tener bloques de código que contengan sentencias continuas, secuenciales ni discretas. Las únicas limitaciones en la conexión de terminales son las definidas por las causalidades de éstos.

3.6.3.- Clases de componentes.

Las clases o componentes son el elemento fundamental en los lenguajes de modelado orientados a objetos. Constituyen la herramienta básica para representar el comportamiento de sistemas simples e indivisibles, o de sistemas muy complejos formados por otros subsistemas.

En ambos lenguajes las clases se definen mediante un conjunto de secciones de código. Cada sección tiene un objetivo determinado en cada uno de los lenguajes. No existe una correspondencia única entre secciones de ambos tipos de lenguaje. En la Tabla 3.1. se muestra qué tipo de código se incluye en cada sección de forma sintética.

Tabla 3.1.: Correspondencia entre secciones.

Declaración de:	**Modelica**	**EL**
Parámetros	declarativa. (‘parameter’)	DATA
Terminales	declarativa	PORTS
Variables protegidas	protected	DECLS
Componentes	declarativa	TOPOLOGY
Conexión entre componentes	equation	TOPOLOGY
Inicialización de variables	initial equation o atributos ‘start’ de variables	INIT
Eventos	equation y algorithm	DISCRETE
Sentencias continuas, ecuaciones	equation	CONTINUOUS
Sentencias secuenciales	algorithm	INIT

En EL, los parámetros se pueden declarar tanto en la sección DATA como en la definición del identificador del componente.

Respecto a la inicialización de las variables en Modelica, las sentencias utilizadas son de carácter declarativo, por lo que la herramienta realizará las manipulaciones algebraicas necesarias para resolver el problema de valor inicial planteado en la simulación. Sin embargo, en EL las inicializaciones se realizan ejecutándose secuencialmente con las instrucciones de inicialización contenidas en el bloque INIT.

En lo que respecta a la declaración de sentencias que definan eventos en EL sólo pueden aparecer en el bloque DISCRETE. En Modelica, la definición de eventos se puede priorizar de forma que ante una posible simultaneidad de eventos siempre se podría tratar los eventos en un orden determinado.

Tanto EL como Modelica muestran ciertas características avanzadas que permiten parametrizar o variar el comportamiento entre clases. En ambos se pueden realizar herencias múltiples y crear clases abstractas. El problema viene cuando en Modelica se pretende crear ecuaciones virtuales o en EL se quieren crear clases paramétricas. Las ecuaciones virtuales son un mecanismo muy flexible para modificar comportamientos de componentes que pertenecen al mismo grafo de herencia. En Modelica esta propiedad no está disponible y cuando una ecuación queda definida en una clase nunca podrá ser modificada en clases descendientes. Esta propiedad es de enorme potencia a la hora de reutilizar librerías pero EL no la contempla.

3.6.4.- Librerías.

Una de las partes fundamentales de este proyecto consiste en crear librerías para EcosimPro, por eso es necesario hablar sobre las pequeñas diferencias que puede haber entre los dos tipos de lenguaje en lo que a creación de librerías se refiere. Para la creación de la librería mecánica rotacional en EL se han tomado como base muchos modelos mecánicos ya creados en Modelica. Las librerías son conceptualmente similares en ambos lenguajes y su objetivo es el de almacenar modelos o clases ya desarrolladas para su posterior utilización.

Una diferencia, que hoy en día ya es menos significativa pero que fue muy importante cuando se creó EL, es la inclusión de propiedades de representación gráfica de los componentes dentro de ellos mismos y en las librerías en los que se almacena. Modelica permite asociar a un componente un icono gráfico dentro de su misma definición. De esta forma, un usuario podría programar la información gráfica de su componente además de modelar su comportamiento. Un diseñador de librerías podría incluso programar directamente en Modelica las primitivas gráficas que representan a la clase en un diagrama gráfico. Normalmente, la descripción gráfica del componente la realiza la herramienta de simulación mediante capturadotes de esquemas. Antiguamente en EcosimPro no era así, en su lugar se utilizaba una herramienta externa llamada *SmartSketch* que proporcionaba al usuario las posibilidades gráficas que no ofrecía el propio EcosimPro. Como se ha comentado en el Apartado 3.4., hoy en día EcosimPro posee una herramienta gráfica llamada EcoDiagram, que sustituye a la antigua herramienta *SmartSketch*, para crear componentes gráficamente arrastrando iconos desde una paleta y conectándolos entre sí. Esta herramienta además genera códigos XML (*eXtensible Markup Language*) que son códigos de lenguajes de marcas extensibles usados para estructurar información en un documento o en general en cualquier fichero que contenga texto, como por ejemplo ficheros de configuración de un programa o una tabla de datos [ATW09].

3.6.5.- Particiones y definición de experimentos.

La partición es un concepto implementado solamente en EcosimPro que permite dar una gran flexibilidad al diseño de experimentos y simulaciones. Es una característica implementada en la herramienta y no en el lenguaje. Ninguna de las posibilidades que ofrece EcosimPro/EcosimPro Language con las particiones se pueden utilizar con Dymola/Modelica.

EL ofrece explícitamente la posibilidad de definir experimentos sobre los sistemas a simular mediante bloques de códigos secuencial, que permite: declarar variables, inicializar variables algebraicas en los modelos, definir condiciones de contorno, cálculo de estacionarios, selección de métodos de integración, selección de tolerancias, generación de informes, invocación de funciones externas, etc. Todas estas posibilidades a la hora de realizarlas con Modelica dependen en gran medida de la herramienta con la que se trabaje. En el caso de Dymola, estas funcionalidades que se obtienen en EL con experimentos se pueden alcanzar, pero de manera más limitada.

En general, ambos lenguajes están perfectamente dotados de construcciones que permiten abordar la metodología de modelado orientado a objetos de sistemas continuos, discretos e híbridos.

Algunos de los puntos que podría mejorar EcosimPro Language para que fuera más cómodo trabajar con él serían:

- Aumentar la capacidad de un "*array*" para contener componentes, además de tipos simples y derivados.

- Introducción de las clases paramétricas.

- Mejorar las prestaciones de la herramienta EcoDiagram para crear componentes gráficamente.

CAPÍTULO 4

4. CREACIÓN DE LIBRERIAS CON ECOSIMPRO

En el presente capítulo del proyecto final de carrera se va a describir la forma de crear una librería con diferentes elementos mecánicos rotacionales. Una de las ventajas de EcosimPro es poder organizar los componentes en librerías dependiendo de su tipo. Se pueden crear librerías con elementos mecánicos, eléctricos, de control, frigoríficos, etc., e incluso como se verá a continuación en este capítulo, se puede crear dentro de la parte mecánica, una librería de elementos mecánicos traslacionales y una librería de elementos mecánicos rotacionales para diferenciar estos dos conceptos.

El fin de una librería es poder agrupar varios elementos que poseen características comunes en un mismo entorno. Para crear una librería hay que modelar una serie de puertos y de clases abstractas que luego ayudaran, mediante la abstracción y la herencia, a poder ir formando los componentes sin la necesidad de repetir constantes ni variables en sus modelos. Las librerías permiten crear sistemas de componentes de una manera mucho más sencilla que modelar directamente el sistema, incluso como ya se ha visto, es posible hacerlo de una manera gráfica usando los símbolos previamente creados de los componentes, la cual es mucho más intuitiva y dinámica que la convencional.

Se han de crear los puertos que se quiere incluir en la librería mecánica rotacional. También hay que crear las clases abstractas de las cuales luego heredan otras clases o componentes. Y hay que definir todos los componentes mecánicos rotacionales de la librería con los cuales se pueden formar sistemas mecánicos rotacionales complejos que más adelante serán objeto de simulaciones y experimentos.

4.1.- Elementos de la librería Rotational Library.

La librería que ha sido creada en EcosimPro está compuesta por elementos mecánicos rotacionales en su mayoría, pero también ha sido necesario añadir algún componente de otro tipo, como son los puertos analógicos, que van a ayudar a poder crear y simular otros componentes o sistemas. Esta librería se llama *Rotational Library* y está dividida en subclases formadas por elementos que comparten características comunes.

En una primera subclase *PORTS* se han añadido los puertos que van a servir de unión entre componentes. En la subclase *COMMON* se pueden encontrar todos los componentes abstractos que por sí solos no tienen significado real pero que harán más

sencilla la creación de otros componentes en diferentes subclases. Las subclases *CONSTRAINS* y *ACTUATORS* están compuestas por componentes que imponen restricciones dinámicas al sistema o componentes que lo excitan respectivamente. En la subclase *BASIC* se pueden encontrar los componentes mecánicos básicos con movimiento rotacional. *BEARING* se compone de un modelo de un cojinete con lubricación hidrodinámica. Y en la última subclase *GEARS* se han añadido los engranajes, componentes modelados para simular efectos característicos del funcionamiento de engranajes, como son el rendimiento o el huelgo entre dientes, y las cajas de cambios usadas en las simulaciones posteriores.

A continuación, se van a agrupar en un esquema todos los componentes creados en el marco de este proyecto fin de carrera, estructurados en subclases, donde se pueden ver las relaciones que hay entre ellos. Ha sido necesario añadir a la librería *Rotacional Library* componentes que pertenecen a librerías propias de EcosimPro, ya que serán necesarios en algún momento de las simulaciones, estos componentes se identifican en la Figura 4.1.

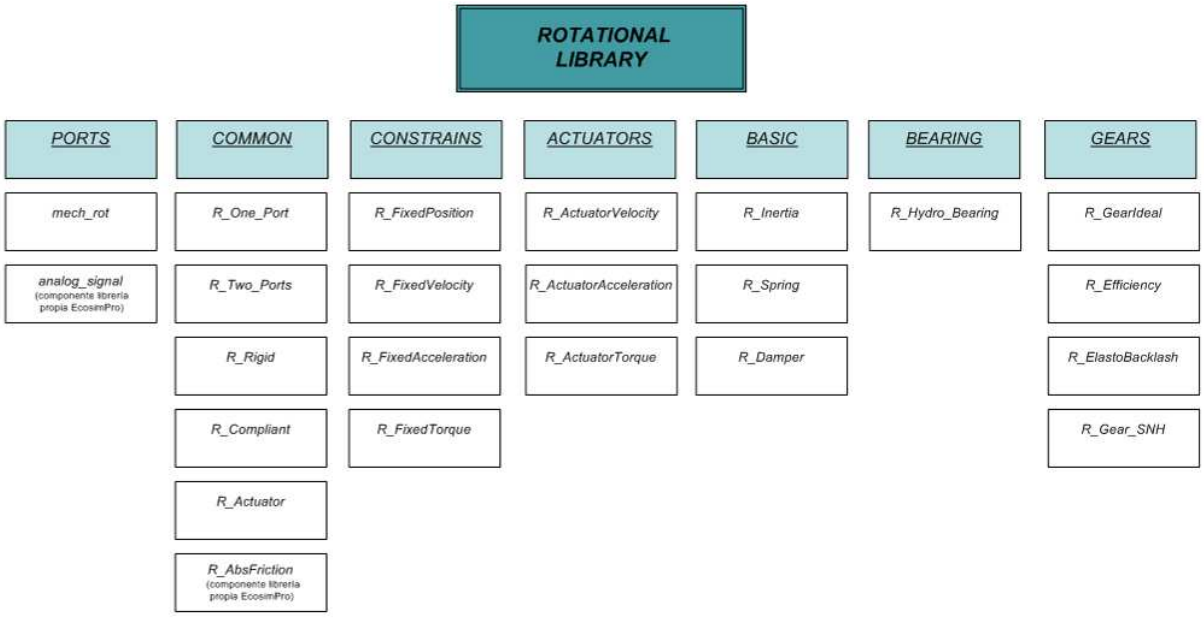


Fig. 4.1.: Estructura de la librería Rotational Library.

En la Tabla 4.1. se recogen todos los componentes creados en la librería *Rotational Library* y se añade a qué subclase pertenecen, qué tipo de componente son y una pequeña descripción de qué es cada uno.

Tabla 4.1.: Elementos de la librería Rotational Library.

Subclase	Nombre	Tipo	Descripción
PORTS	*mech_rot*	Puerto	Puerto mecánico rotacional
	analog_signal		Puerto para señales analógicas
COMMON	*R_One_Port*	Abstracto	Clase para elementos de 1 puerto
	R_Two_Ports		Clase para elementos de 2 puertos
	R_Rigid		Clase para elementos con dos puertos sin movimiento relativos entre ellos
	R_Compliant		Clase para elementos con dos puertos con movimiento relativos entre ellos
	R_Actuator		Clase abstracta que define un actuador genérico rotacional
	R_AbsFriction		Clase abstracta que representa la fricción de Coulomb
CONSTRAINS	*R_FixedPosition*	Concreto	Elemento que impone una posición angular determinada
	R_FixedVelocity		Elemento que impone una velocidad angular determinada
	R_FixedAcceleration		Elemento que impone una aceleración angular determinada
	R_FixedTorque		Elemento que impone un par determinado
BASIC	*R_Inertia*		Disco de inercia
	R_Spring		Muelle rotacional
	R_Damper		Amortiguador rotacional
ACTUATORS	*R_ActuatorVelocity*		Actuador de velocidad
	R_ActuatorTorque		Actuador de par
	R_ActuatorAcceleration		Actuador de aceleración
BEARING	*R_Hydro_Bearing*		Elemento que modela un cojinete con lubricación hidrodinámica
GEARS	*R_GearIdeal*		Elemento que modela la relación de transmisión ideal
	R_Efficiency		Modelo que representa la eficiencia o rendimiento en un sistema mecánico
	R_ElastoBacklash		Sistema anti-huelgo con elasticidad y amortiguación
	R_Gear_SNH		Caja de cambios con fricción hidrodinámica

En los siguientes apartados del presente capítulo se va a explicar en detalle cómo se han creado y cómo funcionan todos los componentes de la Tabla 4.1., además de qué relaciones existen entre ellos, es decir, cómo heredan unos de otros.

4.2.- Relaciones entre componentes.

La herencia es una propiedad de modelado orientado a objetos que se utiliza constantemente en EcosimPro. Como ya se ha comentado en el Apartado 2.2., los componentes heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. La herencia permite a los componentes ser definidos y creados como tipos especializados de objetos preexistentes, estos pueden compartir y extender su comportamiento sin tener que volver a ser implementados.

En la construcción de la librería *Rotational Library* se ha usado esta propiedad de heredar características de clases abstractas en muchos componentes. Para mostrar qué componentes heredan de otros se van a agrupar en el diagrama de la Figura 4.2. donde se aprecia claramente las relaciones que hay entre todos los elementos creados en la librería que se basan en esta propiedad. Por una parte, se tienen los componentes que heredan de la clase abstracta que posee con un puerto mecánico rotacional, *R_One_Port*. Por otra parte, están los componentes que heredan de la clase abstracta que tiene dos puertos mecánicos rotacionales, *R_Two_Ports*, que a su vez se dividen en los que tienen movimiento relativo entre sus puertos, *R_Compliant*, y los que no, *R_Rigid*. Y por último, aparecen los componentes que heredan de la clase abstracta *R_Actuator*.

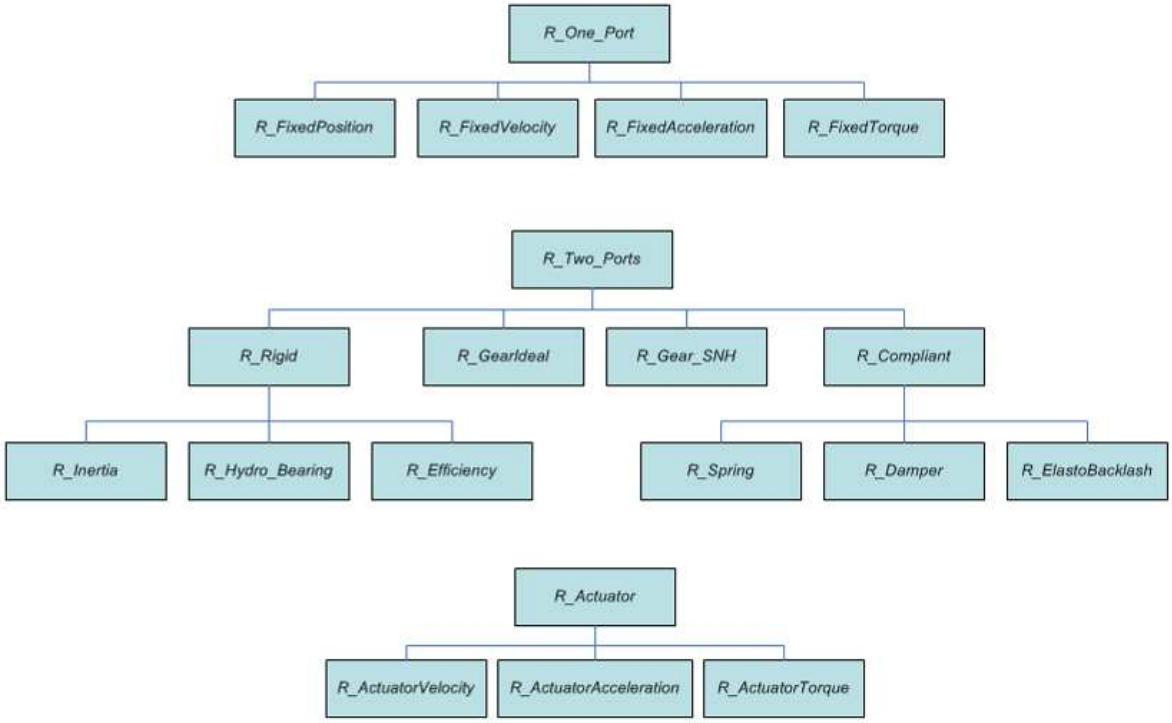


Fig. 4.2.: Herencia entre componentes de Rotational Library.

4.3.- Creación de puertos.

Para modelar un sistema de una forma modular hay que conectar los componentes que forman el sistema usando sus puertos de entrada y salida. Al ser la unión de elementos entre sí necesaria, todos los componentes necesitan en sus modelos unos conectores llamados puertos que fueron analizados en profundidad en el Capítulo 3. La librería *Rotational Library*, consta de dos tipos de puertos, uno de tipo mecánico rotacional llamado *mech_rot* y otro que se utiliza para transmitir las señales emitidas por un componente generador de señales denominado *AnalogSource* al que se llamará *analog_signal*.

4.3.1.- Puerto *mech_rot*.

La conexión de elementos mecánicos rotacionales tiene que hacerse a través de puertos que tengan como variables magnitudes de tipo mecánico rotacional. Esto quiere decir que al tener movimientos rotacionales se debe medir la distancia, por ejemplo, en radianes, pero si se hablara de movimientos traslacionales se debería medir en metros. La Tabla 4.2. muestra una breve descripción y la representación gráfica adoptada para este tipo de puerto.

Tabla 4.2.: Descripción y representación gráfica de mech_rot.

Puerto	Descripción	Representación gráfica
mech_rot	El puerto de entrada o salida principal es un puerto mecánico rotacional	

Las variables que se definen en este puerto son las que se muestran en la Tabla 4.3., como se puede comprobar, son variables que definen movimientos rotacionales.

Tabla 4.3.: Variables definidas en mech_rot.

Variable	Tipo	Descripción	Unidades
tau	REAL	Define el par o momento rotacional	N·m
phi	REAL	Define el ángulo de giro	rad

El código en lenguaje de programación EL que representa el modelo del puerto mecánico rotacional es el de la Figura 4.3.

```
-------------------------------------------------------------------------------
-- Port mech_rot (Rotational port)
-------------------------------------------------------------------------------
PORT mech_rot "1D rotational flange"
   SUM   REAL  tau   UNITS   "N*m"     "Torque (N*m)"
   EQUAL REAL  phi   UNITS   "rad"     "Absolute angular position (rad)"

END PORT
```

Fig. 4.3.: Código del puerto mech_rot.

4.3.2.- Puerto *analog_signal.*

Para transmitir las señales emitidas por el componente generador de señales denominado *AnalogSource*, se ha utilizado el puerto *analog_signal*. La descripción y representación de este componente se muestra en la Tabla 4.4.

Tabla 4.4.: Descripción y representación gráfica de analog_signal.

Puerto	Descripción	Representación gráfica
analog_signal	Componente encargado de transmitir las señales analógicas emitidas por otros componentes	

Las variables que se definen en este puerto son las que se muestran en la Tabla 4.5.

Tabla 4.5.: Variables definidas en analog_signal.

Variable	Tipo	Descripción	Unidades
signal [n]	EQUAL OUT REAL	Define la señal analógica	-

La variable tipo EQUAL OUT específica que es posible conectar un puerto de salida con múltiples puertos de entrada y el valor de la señal en todos estos puertos conectados es el mismo. Además, su código (que ha sido tomado de las librerías que incluye EcosimPro) se presenta en la Figura 4.4.

```
-------------------------------------------------------------------------------
-- Ports analog_signal and bool_signal
-------------------------------------------------------------------------------
PORT analog_signal (INTEGER n = 1 "Number of outputs (-)")   SINGLE IN  "Analog signals 1D port"
   EQUAL OUT REAL signal[n]       "Analog signal values (-)"

END PORT
```

Fig. 4.4.: Código del puerto analog_signal.

La palabra SINGLE IN dentro del modelo quiere decir que la conexión de múltiples entradas en el puerto están prohibidas. Lo que no prohíbe esta palabra es conectar múltiples salidas.

4.4.- Creación de clases abstractas.

Para hacer más sencilla la programación de los componentes se van a crear una serie de clases abstractas las cuales no representan componentes físicos en sí pero son usadas para crear, a partir de ellas, componentes concretos. Estas clases no poseen representación gráfica ya que no representan componentes que luego se vayan a utilizar para generar sistemas.

Estas clases abstractas no se crean de manera arbitraria, sino que surgen debido a que dos o más de los componentes que integrarán la librería poseen características comunes que pueden ser implementadas en estos elementos abstractos, de forma que ahorren numerosas líneas de programa.

Dentro de la librería *Rotational Library*, existen varias clases abstractas. Las clases abstractas *R_One_Port y R_Two_Ports* se utilizan para definir componentes que se construyen con uno y dos puertos mecánicos rotacionales respectivamente, la clase *R_Rigid* se emplea para reproducir las características de los componentes con comportamiento de sólido rígido, la clase *R_Compliant* define la conexión entre dos puertos rotacionales cuando puede existir movimiento relativo entre ellos, la clase *R_Actuator* reúne las características comunes de todos los componentes que funcionan como actuadores y la clase *R_AbsFriction* define la fricción de Coulomb para componentes de dinámica rotacional.

4.4.1.- Clase abstracta *R_One_Port.*

Todos los componentes de la librería *Rotational Library* que precisen únicamente de un puerto mecánico rotacional, podrán heredar las características de esta clase abstracta formada por un sólo puerto, cuya descripción se muestra en la Tabla 4.6.

Tabla 4.6.: Descripción de R_One_Port.

Nombre	Descripción
R_One_Port	Componente abstracto que define elementos con un sólo puerto mecánico rotacional

El puerto de este elemento se presenta en la Tabla 4.7.

Tabla 4.7.: Descripción de los puertos de R_One_Port.

Puerto	Tipo	Dirección
m_out	*mech_rot*	OUT

En esta clase abstracta no aparece ninguna variable, esto no será un problema ya que las variables se pueden definir en el modelo del componente que herede de esta clase. El código en lenguaje de programación EL que representa el modelo del componente abstracto se muestra en la Figura 4.5.

```
--------------------------------------------------------------------------------
-- Component R_One_Port
--------------------------------------------------------------------------------
ABSTRACT COMPONENT R_One_Port
"Abstract class for definition of components with one rotational port"
   PORTS
      OUT mech_rot m_out    "Right / driven mech_rot"

END COMPONENT
```

Fig. 4.5.: Código de la clase abstracta R_One_Port.

4.4.2.- Clase abstracta *R_Two_Ports.*

Esta clase abstracta será la base de todos aquellos elementos que se construyan con dos conectores mecánicos rotacionales, uno de entrada y otro de salida. Su descripción se muestra en la Tabla 4.8.

Tabla 4.8.: Descripción de R_Two_Ports.

Nombre	Descripción
R_Two_Port	Componente abstracto que define elementos con dos puertos mecánicos rotacionales

Esta clase abstracta contiene dos puertos, el de entrada (*m_in*) y el de salida (*m_out*), cuya descripción se presenta en la Tabla 4.9.

Tabla 4.9.: Descripción de los puertos de R_Two_Ports.

Puerto	Tipo	Dirección
m_in	*mech_rot*	IN
m_out	*mech_rot*	OUT

En esta clase abstracta tampoco se declarará ninguna variable, más adelante se verá que las variables se pueden definir en el componente directamente. El código en lenguaje de programación EL para este componente abstracto se representa en la Figura 4.6.

```
-------------------------------------------------------------------------------
-- Component R_Two_Ports
-------------------------------------------------------------------------------
ABSTRACT COMPONENT R_Two_Ports
"Abstract class for definition of components with two rotational ports"
   PORTS
      IN  mech_rot m_in     "Left / driving mech_rot"
      OUT mech_rot m_out    "Right / driven mech_rot"

END COMPONENT
```

Fig. 4.6.: Código de la clase abstracta R_Two_Ports.

4.4.3.- Clase abstracta *R_Rigid*.

Ya que dentro de la clase abstracta *R_Rigid* va a haber componentes que poseen dos conectores, uno de entrada y otro de salida, esta clase puede heredar las propiedades de otra clase abstracta vista con antelación, *R_Two_Ports*. Muchos componentes mecánicos presentan un comportamiento de sólido rígido, es decir, son componentes que están formados por un conjunto de puntos que se mueven de tal manera que no se alteran las distancias entre ellos sea cual sea la fuerza actuante. Los componentes que cumplen con esta propiedad y poseen una única entrada y una única salida deben heredar de la clase abstracta *R_Rigid*, cuya descripción se muestra en la Tabla 4.10.

Tabla 4.10.: Descripción de R_Rigid.

Nombre	Descripción
R_Rigid	Componente abstracto que define elementos con dos puertos mecánicos rotacionales sin movimiento relativo entre ellos

Los puertos de *R_Rigid* se presentan en la Tabla 4.11.

Tabla 4.11.: Descripción de los puertos de R_Rigid.

Puerto	Tipo	Dirección
m_in	*mech_rot*	IN
m_out	*mech_rot*	OUT

El ángulo de giro absoluto *phi*, es la variable que se usa para definir el comportamiento de este tipo de componentes. Las variables que introduce el elemento *R_Rigid* se presentan en la Tabla 4.12.

Tabla 4.12.: Variables que se introducen con R_Rigid.

Variable	Tipo	Descripción	Unidades
phi	REAL	Define el ángulo de giro absoluto	rad

Para definir el comportamiento de los sólidos rígidos es necesario acudir a la fórmula que los representa descrita en la Ecuación (4.1.). Si se quiere modelar el comportamiento de un sólido rígido, hay que imponer que el giro absoluto de los conectores del componente debe ser el mismo, es decir, el ángulo de giro relativo entre la entrada y la salida debe ser nulo:

$$m\_in.phi = m\_out.phi \tag{4.1.}$$

O lo que es lo mismo, representado en las Ecuaciones (4.2.) y (4.3.):

$$m\_in.phi = phi \tag{4.2.}$$

$$m\_out.phi = phi \tag{4.3.}$$

El código en lenguaje de programación EL del componente abstracto *R_Rigid* se representa en la Figura 4.7. Es importante destacar el uso del comando IS_A dentro de este modelo, mediante la que se consigue que la clase abstracta *R_Rigid* herede las características de *R_Two_Ports.*

```
-------------------------------------------------------------------------------
-- Component R_Rigid
-------------------------------------------------------------------------------
ABSTRACT COMPONENT R_Rigid IS_A R_Two_Ports
"Abstract class for the definition of a rigid connection of two rotational ports"
   DECLS
      REAL phi   UNITS   "rad"   "Absolute angular position (rad)"

   CONTINUOUS
      m_in.phi = phi
      m_out.phi = phi

END COMPONENT
```

Fig. 4.7.: Código de la clase abstracta R_Rigid.

4.4.4.- Clase abstracta *R_Compliant.*

Esta clase abstracta, cuya descripción se muestra en la Tabla 4.13., se utiliza para modelar aquellos elementos que poseen dos conectores rotacionales con movimiento relativo entre ellos. Además, el valor del par en ambos conectores o puertos debe ser el mismo. Al igual que ocurría con *R_Rigid*, en este caso también se heredan las propiedades de *R_Two_Ports* vía IS_A, ya que esta clase abstracta también posee dos conectores, uno de entrada y otro de salida.

Tabla 4.13.: Descripción de R_Compliant.

Nombre	Descripción
R_Compliant	Componente abstracto que define elementos con dos puertos mecánicos rotacionales con movimiento relativo entre ellos

Los puertos que posee este elemento se presentan en la Tabla 4.14.

Tabla 4.14.: Descripción de los puertos de R_Compliant.

Puerto	Tipo	Dirección
m_in	*mech_rot*	IN
m_out	*mech_rot*	OUT

Para modelar la clase *R_Compliant* es necesario introducir la constante giro relativo inicial *phi_rel_0* así como las variables ángulo de giro relativo *phi_rel* y par *tau.* Esto se hace en las Tablas 4.15. y 4.16. respectivamente.

Tabla 4.15.: Constantes que se introducen con R_Compliant.

Dato	Tipo	Descripción	Valor inicial	Unidades
phi_rel_0	REAL	Define la distancia angular inicial entre los dos puertos	0	rad

Tabla 4.16.: Variables que se introducen con R_Compliant.

Variable	Tipo	Descripción	Unidades
tau	REAL	Define el par o momento rotacional	N·m
phi_rel	REAL	Define el ángulo de giro relativo	rad

Hay ocasiones en las cuales para poder obtener el modelo de algún componente es necesario dar un valor inicial a alguna de sus variables. Esto ocurre cuando hay más incógnitas que ecuaciones y resulta imposible hallar todas las incógnitas. Cuando esto sucede es necesario inicializar alguna de las variables con una ecuación de contorno o asignarle un valor inicial. Las ecuaciones de contorno definen la forma en la que actúa el elemento en un momento dado. Para el caso de *R_Compliant* se introduce la Ecuación (4.4.), condición de contorno que expresa que en el estado inicial el ángulo de giro relativo coincide con el ángulo de giro absoluto.

$$phi\_rel = phi\_rel\_0 \tag{4.4.}$$

Para definir el giro relativo entre los dos puertos se utiliza la Ecuación (4.5.), en la que se calcula el valor de la variable *phi_rel*. Mientras que la Ecuación (4.6.) indica que el valor del par *tau* ha de ser el mismo en los dos extremos del componente:

$$phi\_rel = m\_out.phi - m\_in.phi \tag{4.5.}$$

$$m\_in.tau = m\_out.tau \tag{4.6.}$$

El código en lenguaje de programación EL que representa la clase abstracta *R_Compliant* se representa en la Figura 4.8., la Ecuación (4.6.) no puede ser escrita directamente en el modelo debido a que es necesario escribir las ecuaciones de manera que aparezca siempre una variable o constante, es decir, no se puede escribir una ecuación en la que se iguale el valor de un puerto al valor de otro puerto.

```
-------------------------------------------------------------------------------
-- Component R_Compliant
-------------------------------------------------------------------------------
ABSTRACT COMPONENT R_Compliant IS_A R_Two_Ports
"Abstract class for definition of a compliant connection of two rotational ports"
   DATA
      REAL phi_rel_0 = 0   UNITS "rad"   "Initial angular distance between ports (rad)"

   DECLS
      REAL tau       UNITS "N*m"       "Rotational transmited torque (N*m)"
      REAL phi_rel   UNITS "rad"       "Angular distance between ports (rad)"

   INIT
      phi_rel = phi_rel_0

   CONTINUOUS
      m_in.tau = tau
      m_out.tau = tau

      phi_rel = m_out.phi - m_in.phi

END COMPONENT
```

Fig. 4.8.: Código de la clase abstracta R_Compliant.

4.4.5.- Clase abstracta *R_Actuator*.

La clase abstracta *R_Actuator*, cuya descripción se muestra en la Tabla 4.17., constituye una clase abstracta que se utiliza para definir actuadores rotacionales genéricos. Está formado por dos puertos, uno de entrada de tipo *analog_signal* y otro de salida de tipo *mech_rot* descritos en la Tabla 4.18.

Tabla 4.17.: Descripción de R_Actuator.

Nombre	Descripción
R_Actuator	Clase abstracta que se utiliza para definir actuadores rotacionales genéricos

Tabla 4.18.: Descripción de los puertos de R_Actuator.

Puerto	Tipo	Dirección
s_in	*analog_signal*	IN
m_out	*mech_rot*	OUT

En esta clase abstracta no se va a definir ninguna constante o variable. El código en lenguaje de programación EL que representa el componente abstracto *R_Actuator* se muestra en la Figura 4.9.

```
-------------------------------------------------------------------------------
-- Component R_Actuator
-------------------------------------------------------------------------------
ABSTRACT COMPONENT R_Actuator
"Abstract class for definition of generic rotational actuators"
   PORTS
      IN  analog_signal s_in     "Input signal port"
      OUT mech_rot m_out         "Outlet rotational mechanical port"

END COMPONENT
```

Fig. 4.9.: Código de la clase abstracta R_Actuator.

4.4.6.- Clase abstracta *R_AbsFriciton.*

Mediante el componente abstracto *R_AbsFriction* se consigue modelar el fenómeno de la fricción. Este fenómeno es vital para la simulación de sistemas mecánicos, ya que gracias a su existencia se puede explicar el comportamiento de aquellos dispositivos con movimiento relativo entre superficies que están en contacto. La fricción presenta un marcado comportamiento discontinuo, especialmente para bajas velocidades relativas entre las superficies en contacto, lo que complica su modelado [OTT99].

Se han desarrollado multitud de modelos con el propósito de modelar la fricción, aquí se van clasificar en dos grandes grupos: los de tipo discontinuo y los de tipo continuo. En los modelos de tipo discontinuo la fricción (par de fricción) es discontinua para velocidad nula y cuando esto ocurre (modo estático) el par de fricción actúa equilibrando el resto de los pares aplicados al sistema para mantener la velocidad nula. Por otro lado, los modelos continuos consideran pequeños desplazamientos elásticos (desplazamientos de pre-deslizamiento) en el modo estático.

Para modelar la fricción en la librería *Rotational Library* se ha elegido un modelo de tipo discontinuo, debido a la intuitiva simplicidad que éstos presentan. A pesar de esta ventaja, los modelos discontinuos siguen resultando incómodos ya que su definición para velocidad nula y no nula es totalmente distinta. Una forma de afrontar este inconveniente consiste en utilizar un umbral de velocidad que funcione como frontera entre las regiones de velocidad nula y no nula. Sin embargo, la elección de este umbral tiene una gran influencia en el comportamiento de los modelos, y una mala elección del mismo puede dar lugar a comportamientos no deseados. Otra forma de afrontar el problema de la discontinuidad, y que es la que utiliza el componente *R_AbsFriction*, consiste en emplear una máquina de estados finitos basada en la detección de cambios de signo de la velocidad [KIK05].

En definitiva, la clase abstracta *R_AbsFriction* está basada en un modelo de tipo híbrido y dispone de una máquina de estados finitos que detecta los cambios de signo en la velocidad. La Figura 4.10. representa un eje que gira a una velocidad angular *w(t)* debido a la acción del par *u(t)* y genera un rozamiento *tau_friction*.

Fig. 4.10.: Sistema con fricción de Coulomb.

Aquí, el par de fricción *tau_friction* actúa entre las superficies del eje y su alojamiento, y es una función lineal de la velocidad angular relativa *w_relfric* existente entre ellas, cuando deslizan una sobre otra. Cuando la velocidad relativa se vuelve nula, las dos superficies se encuentran adheridas y el par de fricción ya no es una función de *w_relfric*. El elemento comienza a girar de nuevo si el par *u(t)* supera el valor del par de fricción estático *tau0*. El sistema de la Figura 4.10., que representa uno de los casos más simples de fricción en elementos con movimiento rotacional, se puede representar mediante una curva parametrizada como la de la Figura 4.11., de acuerdo con los sistemas de Ecuaciones (4.7.) y (4.8.), donde además de los términos *w_relfric*, *tau_friction* y *tau0*, que fueron introducidos con anterioridad, aparece el término *s* que representa el parámetro de la curva y *tau_pos* que representa el par de fricción dinámico. La parametrización consiste en la caracterización de un sistema, generalmente discontinuo, a través de una curva dependiente de un parámetro. Esta caracterización describe el sistema de una forma más general, lo que permite describir el sistema de un modo declarativo:

$$w_{relfric} = \begin{cases} s-1 & si \quad s > 1 \\ s+1 & si \quad s < -1 \\ 0 & resto \end{cases} \tag{4.7.}$$

$$tau_{friction} = \begin{cases} tau0 + tau_{pos} * (s-1) & si \quad s > 1 \\ -tau0 + tau_{pos} * (s+1) & si \quad s < -1 \\ tau0 * s & resto \end{cases} \tag{4.8.}$$

Mediante el parámetro *s*, se consigue parametrizar el comportamiento del sistema de la Figura 4.10. y así, las variables del sistema ya no dependen del tiempo sino de *s*.

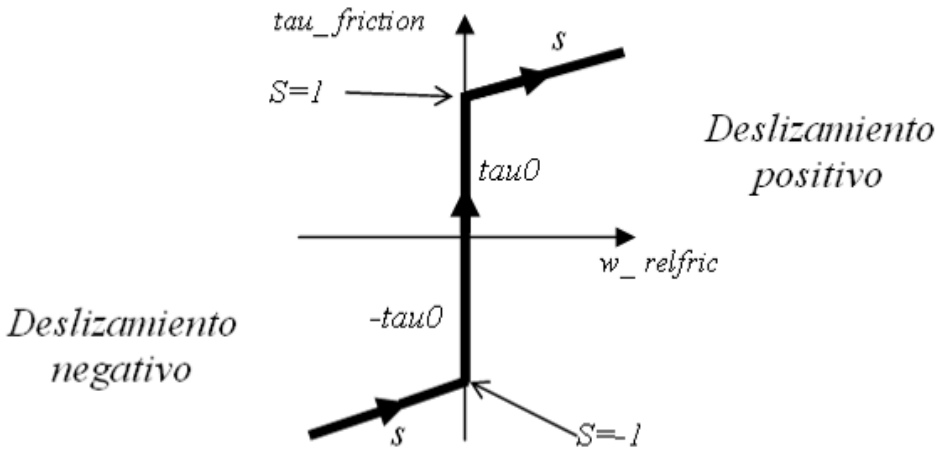


Fig. 4.11.: Curva parametrizada6 de un sistema con fricción de Coulomb.

En la curva parametrizada de la Figura 4.11., en función del valor del parámetro *s* se pueden tener tres modos de operación diferentes. El modo *estático*, para *-1≤s≤1;* el modo de *deslizamiento positivo*, para *s>1*; y el modo de *deslizamiento negativo*, para *s<-1*.

El modelo matemático implementado por los sistemas de Ecuaciones (4.7.) y (4.8.), describe por completo el fenómeno de la fricción de una forma declarativa, en un sistema muy simplificado. Por desgracia, debido a la dificultad que entraña definir con precisión la región de velocidad nula y sus proximidades, resulta complicado transformar el modelo de los sistemas de Ecuaciones (4.7.) y (4.8.) en otro que pueda ser simulado. Para comprender estas dificultades se analizará el sistema formado por un eje que gira en un alojamiento de la Figura 4.10., cuyo comportamiento puede caracterizarse mediante la Ecuación (4.9.):

$$I \cdot \dot{w}_{relfric} = u(t) - tau_{friction} \tag{4.9.}$$

En la Ecuación (4.9.), *I* es la inercia del eje y *u(t)* es el par conductor, que es un dato. Si el elemento se encuentra en el modo *deslizamiento positivo*, entonces *s ≥1*, y por lo tanto el modelo queda descrito por las Ecuaciones (4.9.), (4.10.) y (4.11.), y se puede transformar fácilmente en un espacio de estados, con *w_relfric* como variable de estado:

$$tau_{friction} = tau0 - tau_{pos} \cdot (s-1) \tag{4.10.}$$

$$w_{relfric} = s - 1 \tag{4.11.}$$

Así, sí el eje se detiene, entonces *-1≤ s ≤ 1* y el valor *w_relfric* pasa a ser nulo, por lo tanto *w_relfric* no puede ser un estado, lo que origina un cambio de índice de las ecuaciones diferenciales. Pero junto con la dificultad de manejar los cambios en las variables de estado, existe otro problema más serio si se supone que el eje se encuentra estático y el valor del parámetro *s* se vuelve mayor que la unidad. Justo antes de que esto ocurra se tienen las condiciones de las Ecuaciones (4.12.) y (4.13.):

$$s \leq 1 \tag{4.12.}$$

$$w_{relfric} = 0 \tag{4.13.}$$

Pero desde el momento en que se cumple que *s>1* (ver Figura 4.11.) el sistema pasa al modo de *deslizamiento positivo* donde *w_relfric* es una variable de estado, que es inicializada con su último valor *w_relfric=0*, y *s* es calculada a partir de ella según la Ecuación (4.11.), de modo que se obtiene un valor para este parámetro igual a la unidad, resultando así la relación *s>1* falsa, lo que provoca que el sistema vuelva a pasar al modo *estático*. En otras palabras, nunca será posible pasar al modo de *deslizamiento positivo*.

La clave para solucionar este problema consiste en observar que *w_relfric* es nula en el modo *estático* y al inicio del modo de *deslizamiento positivo*, pero sin embargo, la aceleración relativa $\dot{w}_{relfric}$ es mayor que cero cuando comienza el *deslizamiento* y nula en el modo *estático*. De manera que para el instante en el que *w_relfric* es nula, se puede obtener una nueva curva parametrizada como la de la Figura 4.12., que necesita de un nuevo parámetro s_a para satisfacer los sistemas de Ecuaciones (4.14.), (4.15.) y (4.16.):

$$a_{relfric} = der(w_{relfric}) \tag{4.14.}$$

$$tau_{friction} = \begin{cases} tau0 & si \quad s_a > 1 \\ -tau0 & si \quad s_a < -1 \\ tau0 \cdot s_a & resto \end{cases} \tag{4.15.}$$

$$a_{relfric} = \begin{cases} s_a - 1 & si \quad s_a > 1 \\ s_a + 1 & si \quad s_a < -1 \\ 0 & resto \end{cases} \tag{4.16.}$$

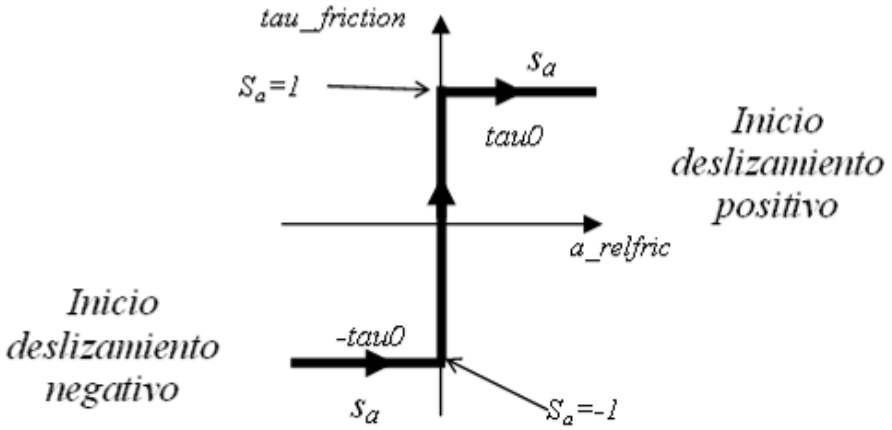


Fig. 4.12.: Curva parametrizada de un sistema con fricción de Coulomb, para w_relfric=0.

Entonces para *w_relfric=0*, los sistemas de Ecuaciones (4.9.), (4.14.), (4.15.) y (4.16.) forman un conjunto de ecuaciones continuas/discretas que tienen que ser resueltas durante todos los instantes de tiempo, con lo que la velocidad *w_relfric* vuelve a ser una variable de estado incluso cuando es nula.

Consecuentemente, si se pasa del modo *estático* a cualquiera de los modos de *deslizamiento*, la velocidad angular relativa será muy pequeña, pero ha de tener algún signo, y será precisamente este signo el que permita conocer si el sistema desliza o no, y si lo hace en sentido positivo o negativo, dependiendo de la ley de la mano derecha [WEI99].

Por tanto, para modelar el fenómeno de la fricción, el componente *R_AbsFriction* se basa en la curva parametrizada de la Figura 4.13., que es idéntica a la de la Figura 4.12. salvo por la existencia de un salto en el par de fricción cuando se inicia el *deslizamiento*.

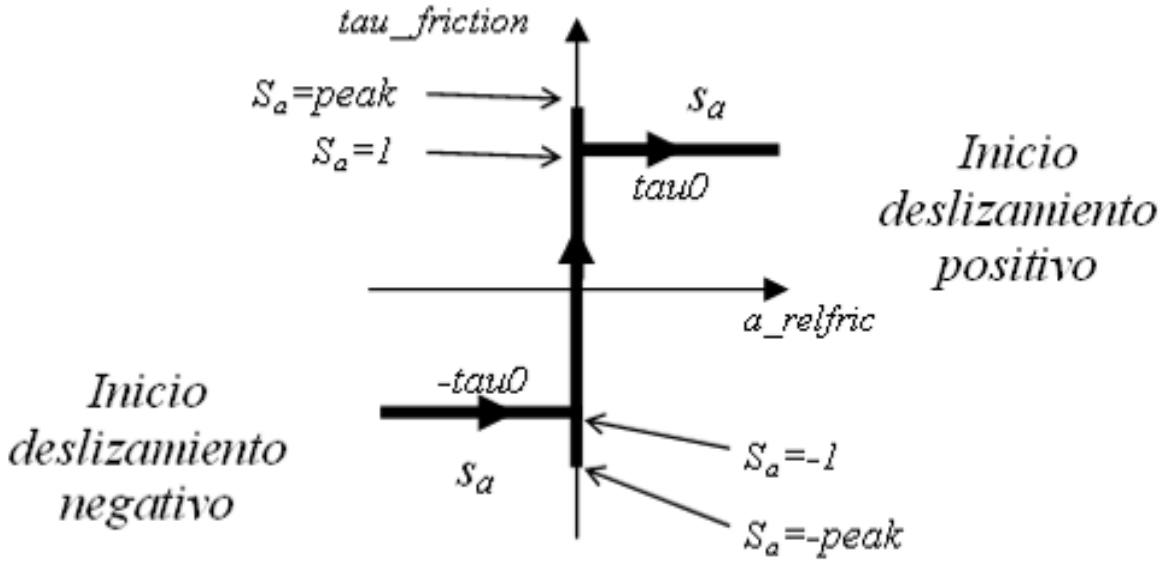


Fig. 4.13.: Curva parametrizada para R_AbsFriction.

Este salto, denominado *peak*, se declarará dentro de los elementos concretos que hereden de *R_AbsFriction*, cuya descripción se muestra en la Tabla 4.19.

Tabla 4.19.: Descripción de R_AbsFriction.

Nombre	Descripción
R_AbsFriction	Clase abstracta que se utiliza para simular la fricción de Coulomb en componente con movimiento rotacional

En la Tabla 4.20. se describen las variables declaradas en el componente abstracto *R_Absfriction.*

Tabla 4.20.: Descripción de las variables declaradas en R_AbsFriction.

Variable	Tipo	Descripción	Unidades
free	BOOLEAN	Variable verdadera si el elemento fricción no está activado y falsa cuando existe fricción	-
w_relfric	REAL	Define la velocidad angular relativa entre las superficies en contacto	rad/s
a_relfric	REAL	Define la aceleración angular relativa entre las superficies en contacto	rad/s^2
tau_friction	REAL	Par de fricción resultante. Se considera positivo si se opone a la velocidad angular relativa	N·m
tau0	REAL	Par de fricción para *w_relfric=0* y para modo de inicio del deslizamiento (*imode 1* ó *imode -1*)	N·m
tau0_max	REAL	Par de fricción estático para *w_relfric=0* (*imode 0*). Depende del valor de *peak*	N·m
sa	REAL	Parámetro de la curva parametrizada para la fricción	-
imode	INTEGER	Modos de operación según el tipo de *deslizamiento*: • 0 si no hay movimiento (w_relfric=a_relfric=0) • 1 si se inicia el *deslizamiento* positivo (w_relfric=0 y a_relfric>0) • -1 si se inicia el *deslizamiento* negativo (w_relfric=0 y a_relfric<0) • 2 si hay *deslizamiento* positivo (w_relfric>0) • -2 si hay *deslizamiento* negativo (w_relfric<0)	-

El código en lenguaje de programación EL que representa la clase abstracta *R_AbsFriction* se representa en la Figura 4.14.

```
-------------------------------------------------------------------------------
-- Component R_AbsFriction
-------------------------------------------------------------------------------
ABSTRACT COMPONENT R_AbsFriction
"Abstract component of rotational coulomb friction components "
   DECLS
      BOOLEAN free = FALSE "TRUE, if frictional element is not active (-)"
      REAL w_relfric       "Relative angular velocity between frictional surfaces (rad/s)"
      REAL a_relfric       "Relative angular acceleration between frictional surfaces (rad/s^2)"
      REAL tau_friction    "Friction torque: positive, if directed in opposite direction of w_rel (N*m)"
      REAL tau0            "Friction torque for w=0 and forward sliding (N*m)"
      REAL tau0_max        "Maximum friction torque for w=0 and locked (N*m)"
      REAL sa              "Path parameter of friction characteristic (-)"
      INTEGER imode = -2   "Operation mode (-)"

   DISCRETE

      WHEN(free) THEN
         imode = 3 AFTER 0.0
      END WHEN

      WHEN (imode == 1 AND w_relfric > 0) THEN
         imode = 2 AFTER 0.0
      END WHEN

      WHEN (imode == 2 AND w_relfric <= 0) THEN
         imode = 0 AFTER 0.0
      END WHEN

      WHEN (imode == -1 AND w_relfric < 0) THEN
         imode = -2 AFTER 0.0
      END WHEN

      WHEN (imode == -2 AND w_relfric >= 0) THEN
         imode = 0 AFTER 0.0
      END WHEN

      WHEN (imode == 0 AND sa > tau0_max) THEN
         imode = 1 AFTER 0.0
      END WHEN

      WHEN (imode == 0 AND sa < -tau0_max) THEN
         imode = -1 AFTER 0.0
      END WHEN


   CONTINUOUS

      fun_R_AbsFriction(imode, a_relfric, sa, tau0) = 0

END COMPONENT
```

Fig. 4.14.: Código del componente R_AbsFriction.

Como se puede observar, el código de la Figura 4.14. además de declarar variables que ya se describen en la Tabla 4.20., introduce una máquina de estados como la de la Figura 4.15. para controlar el valor del par de fricción durante los distintos modos de *deslizamiento*.

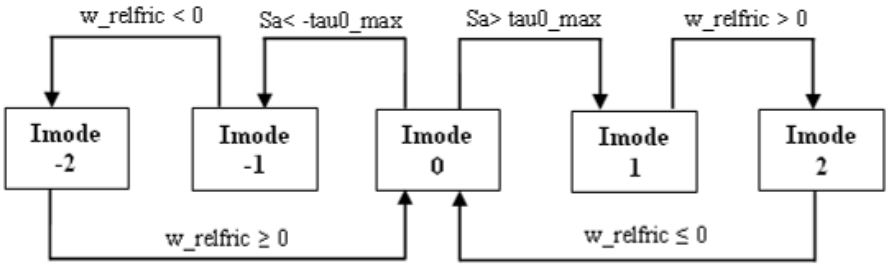


Fig. 4.15.: Máquina de estados de R_AbsFriction.

Además, en el código del modelo de *R_AbsFrcition* aparece una función denominada *fun_R_AbsFriction*, cuyo código aparece en la Figura 4.16., se trata de una función residual creada para calcular implícitamente el valor del parámetro s_a.

```
-------------------------------------------------------------------------------
-- Function fun_R_AbsFriction
-------------------------------------------------------------------------------
FUNCTION REAL fun_R_AbsFriction(
   INTEGER imode  "Operation mode",
   REAL a_relfric "Relative angular acceleration (rad/s^2)",
   REAL sa      "Path parameter of friction characteristic (-)",
   REAL tau0      "Friction torque for w=0 (N*m)")
"Residue function to calculate implicitly the path parameter of friction \
characteristic of the components that inherit from the abstract component R_AbsFriction"

   DECLS
      REAL resul
   BODY
      IF(imode == 0) THEN
         resul = a_relfric
      ELSEIF(imode == 1) THEN
         resul = a_relfric - (sa - tau0)
      ELSEIF(imode == -1) THEN
         resul = a_relfric - (sa + tau0)
      ELSEIF(imode == 2) THEN
         resul = a_relfric - (sa - tau0)
      ELSEIF(imode == -2) THEN
         resul = a_relfric - (sa + tau0)
      ELSE
         resul = a_relfric - sa
      END IF

   RETURN resul

END FUNCTION
```

Fig. 4.16.: Código de la función fun_R_AbsFriction.

4.5.- Creación de componentes.

Una vez creadas las clases abstractas, se van a generar todos los componentes que van a ser usados más adelante en la creación de sistemas mecánicos rotacionales. Puede haber componentes que no necesariamente hereden de una clase abstracta dentro de una librería, este caso ocurre si no comparten ningún elemento en común con otro componente. Se va a estructurar esta parte de la librería *Rotational Library* dividiendo los componentes en las subclases definidas anteriormente, de manera que estén todos los componentes ordenados.

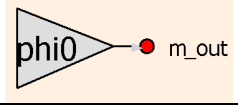
4.5.1.- Componentes de la subclase *CONSTRAINS*.

Los componentes de la subclase *CONSTRAINS* imponen restricciones dinámicas a otros elementos o a una parte del sistema al que pertenezcan.

4.5.1.1.- Componente *R_FixedPosition*.

Este componente hereda de *R_One_port*, por lo tanto toma todas las propiedades de esta clase abstracta. La descripción de *R_FixedPosition* y su representación gráfica se muestra en la Tabla 4.21.

Tabla 4.21.: Descripción y representación gráfica de R_fixedPosition.

Nombre	Descripción	Representación gráfica
R_FixedPosition	Define un elemento que fija la posición angular a la salida del puerto	

El puerto y la constante definida en el elemento *R_FixedPosition* se presentan en las Tablas 4.22. y 4.23. respectivamente.

Tabla 4.22.: Descripción del puerto de R_FixedPosition.

Puerto	Tipo	Dirección
m_out	*mech_rot*	OUT

Tabla 4.23.: Descripción del dato declarado en R_FixedPosition.

Dato	Tipo	Descripción	Valor inicial	Unidades
phi_0	REAL	Define la posición angular fija del puerto	0	rad

Este elemento impone una posición angular fija e igual al valor del parámetro *phi_0*. Así, la Ecuación (4.17.) caracteriza el comportamiento de *R_FixedPosition*:

$$m\_out.phi = phi\_0 \tag{4.17.}$$

El código en lenguaje de programación EL que representa el componente se muestra en la Figura 4.17.

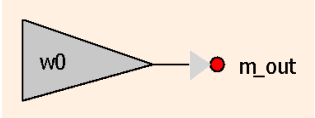
```
-------------------------------------------------------------------------------
-- Component R_FixedPosition
-------------------------------------------------------------------------------
COMPONENT R_FixedPosition IS_A R_One_Port
"Fixed angular position"
   DATA
      REAL phi_0 = 0   UNITS "rad"     "Fixed offset position (rad)"

   CONTINUOUS
      m_out.phi = phi_0

END COMPONENT
```

Fig. 4.17.: Código del componente R_FixedPosition.

4.5.1.2.- Componente *R_FixedVelocity*.

Este componente, al igual que *R_FixedPosition* y el resto de componentes de esta subclase, hereda de *R_One_port*, por lo tanto toma todas las propiedades de esta clase abstracta. En la Tabla 4.24. hay una pequeña descripción y una representación gráfica que representa al componente.

Tabla 4.24.: Descripción y representación gráfica de R_FixedVelocity.

Nombre	Descripción	Representación gráfica
R_FixedVelocity	Define un elemento que fija la velocidad angular a la salida del puerto	

El puerto y la constante que introduce este elemento se presentan en las Tablas 4.25. y 4.26. respectivamente.

Tabla 4.25.: Descripción del puerto de R_FixedVelocity.

Puerto	Tipo	Dirección
m_out	*mech_rot*	OUT

Tabla 4.26.: Descripción del dato declarado en R_FixedVelocity.

Dato	Tipo	Descripción	Valor inicial	Unidades
w0	REAL	Define la velocidad angular fija en el puerto	0	rad/s

La Ecuación (4.18.) representa el comportamiento de *R_FixedVelocity*, al imponer una velocidad angular constante e igual al valor del parámetro *w0:*

$$m\_out.phi' = w0 \qquad (4.18.)$$

El código en lenguaje de programación EL que representa el componente se muestra en la Figura 4.18.

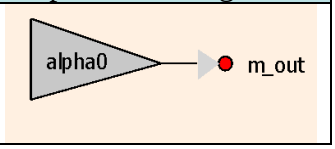
```
-------------------------------------------------------------------------------
-- Component R_FixedVelocity
-------------------------------------------------------------------------------
COMPONENT R_FixedVelocity IS_A R_One_Port
"Fixed angular velocity"
   DATA
      REAL w0 = 0   UNITS "rad/s"   "Fixed angular velocity (rad/s)"

   CONTINUOUS
      m_out.phi' = w0

END COMPONENT
```

Fig. 4.18.: Código del componente R_FixedVelocity.

4.5.1.3.- Componente *R_FixedAcceleration.*

R_FixedAcceleration hereda de *R_One_port,* por lo tanto toma todas las propiedades de esta clase abstracta. Su descripción se muestra en la Tabla 4.27.

Tabla 4.27.: Descripción y representación gráfica de R_FixedAcceleration.

Nombre	Descripción	Representación gráfica
R_FixedAcceleration	Define un elemento que fija la aceleración angular a la salida del puerto	

El puerto y la constante que introduce este elemento se presentan en las Tablas 4.28. y 4.29. respectivamente.

Tabla 4.28.: Descripción del puerto de R_FixedAcceleration.

Puerto	Tipo	Dirección
m_out	*mech_rot*	OUT

Tabla 4.29.: Descripción del dato declarado en R_FixedAcceleration.

Dato	Tipo	Descripción	Valor inicial	Unidades
alpha_0	REAL	Define la aceleración angular fija en el puerto	0	rad/s^2

Este elemento impone una aceleración angular constante e igual al valor del parámetro *alpha_0*. La Ecuación (4.19.) caracteriza el comportamiento de *R_FixedAcceleration*:

$$m\_out.phi'' = alpha\_0 \qquad (4.19.)$$

El código en lenguaje de programación EL que representa el componente se muestra en la Figura 4.19.

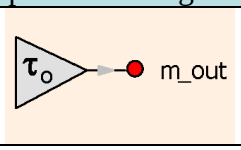
```
-------------------------------------------------------------------------------
-- Component R_FixedAcceleration
-------------------------------------------------------------------------------
COMPONENT R_FixedAcceleration IS_A R_One_Port
"Fixed angular acceleration"
   DATA
      REAL alpha_0 = 0   UNITS "rad/s^2"     "Fixed angular acceleration value (rad/s^2)"

   CONTINUOUS
      m_out.phi'' = alpha_0

END COMPONENT
```

Fig. 4.19.: Código del componente R_FixedAcceleration.

4.5.1.4.- Componente *R_FixedTorque.*

Este componente también hereda de *R_One_port*, por lo tanto toma todas las propiedades de esta clase abstracta. Su descripción y su representación gráfica se muestran en la Tabla 4.30.

Tabla 4.30.: Descripción y representación gráfica de R_FixedTorque.

Nombre	Descripción	Representación gráfica
R_FixedTorque	Define un elemento que fija el par rotacional a la salida del puerto	

El puerto y la constante que introduce este elemento se presentan en las Tablas 4.31. y 4.32. respectivamente.

Tabla 4.31.: Descripción del puerto de R_FixedTorque.

Puerto	Tipo	Dirección
m_out	*mech_rot*	OUT

Tabla 4.32.: Descripción del dato declarado en R_FixedTorque.

Dato	Tipo	Descripción	Valor inicial	Unidades
tau_0	REAL	Define el par fijo del puerto	0	N·m

Este elemento impone un par constante e igual al valor del parámetro *tau_0*. La Ecuación (4.20.) caracteriza el comportamiento de *R_FixedTorque*:

$$m\_out.tau = tau\_0 \qquad (4.20.)$$

El código en lenguaje de programación EL que representa el componente se muestra en la Figura 4.20.

```
-------------------------------------------------------------------------------
-- Component R_FixedTorque
-------------------------------------------------------------------------------
COMPONENT R_FixedTorque IS_A R_One_Port
"Fixed torque"
   DATA
      REAL tau_0 = 0.   UNITS "N*m"    "Fixed torque value (N*m)"

   CONTINUOUS
      m_out.tau = tau_0

END COMPONENT
```

Fig. 4.20.: Código del componente R_FixedTorque.

4.5.2.- Componentes de la subclase *ACTUATORS*.

En esta subclase se encuentran los tres actuadores genéricos de la librería *Rotational Library*. Estos actuadores están diseñados para recibir señales de una fuente analógica y transformarlas en señales que puedan ser interpretadas por los puertos mecánicos rotacionales del resto de los componentes de la librería.

Todos los componentes de esta subclase heredan de la clase abstracta *R_Actuator*, por tanto, estarán formados por un puerto de entrada tipo *analog_signal* y un puerto de salida tipo *mech_rot*, cuya descripción se muestra en la Tabla 4.33.

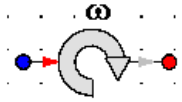
Tabla 4.33.: Descripción de los puertos de los componentes de la subclase R_Actuators.

Puerto	Tipo	Dirección
s_in	*analog_signal*	IN
m_out	*mech_rot*	OUT

4.5.2.1.- Componente *R_ ActuatorVelocity*.

La descripción y la representación gráfica del componente *R_ActuatorVelocity* se muestra en la Tabla 4.34.

Tabla 4.34.: Descripción y representación gráfica de R_ActuatorVelocity.

Nombre	Descripción	Representación gráfica
R_ActuatorVelocity	Define un actuador de velocidad angular	

Este componente transforma la señal de entrada según la Ecuación (4.21.), que proviene de una fuente analógica *s_in.signal[1]*, en una señal de salida que los puertos de tipo *mech_rot* identifican como velocidad angular *m_out.phi'*:

$$m\_out.phi' = s\_in.signal[1] \tag{4.21}$$

El código en lenguaje de programación EL que representa el componente se muestra en la Figura 4.21.

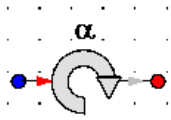
```
-------------------------------------------------------------------------------
-- Component R_ActuatorVelocity
-------------------------------------------------------------------------------
COMPONENT R_ActuatorVelocity IS_A R_Actuator
"Angular velocity actuator"
   CONTINUOUS
      m_out.phi' = s_in.signal[1]

END COMPONENT
```

Fig. 4.21.: Código del componente R_ActuatorVelocity.

4.5.2.2.- Componente *R_ ActuatorAcceleration.*

La descripción y la representación gráfica del componente *R_ActuatorAcceleration* se muestra en la Tabla 4.35.

Tabla 4.35.: Descripción y representación gráfica de R_ActuatorAcceleration.

Nombre	Descripción	Representación gráfica
R_ActuatorAcceleration	Define un actuador de aceleración angular	

Este componente transforma la señal de entrada según la Ecuación (4.22.), que proviene de una fuente analógica *s_in.signal[1]*, en una señal de salida que los puertos de tipo *mech_rot* identifican como una aceleración angular *m_out. phi''*:

$$m\_out.phi'' = s\_in.signal[1] \qquad (4.22.)$$

El código en lenguaje de programación EL que representa el componente se muestra en la Figura 4.22.

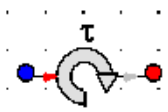
```
-------------------------------------------------------------------------------
-- Component R_ActuatorAcceleration
-------------------------------------------------------------------------------
COMPONENT R_ActuatorAcceleration IS_A R_Actuator
"Rotational acceleration actuator"
 CONTINUOUS
      m_out.phi'' = s_in.signal[1]

END COMPONENT
```

Fig. 4.22.: Código del componente R_ActuatorAcceleration.

4.5.2.3.- Componente *R_ ActuatorTorque.*

La descripción y representación gráfica del componente *R_ActuatorTorque* se muestra en la Tabla 4.36.

Tabla 4.36.: Descripción y representación gráfica de R_ActuatorTorque.

Nombre	Descripción	Representación gráfica
R_ActuatorTorque	Define un actuador de par de torsión	

Este componente transforma la señal de entrada según la Ecuación (4.23.), que proviene de una fuente analógica *s_in.signal[1]*, en una señal de salida que los puertos de tipo *mech_rot* identifican como un par de torsión *m_out.tau*:

$$m\_out.tau = s\_in.signal[1] \qquad (4.23.)$$

El código en lenguaje de programación EL que representa el componente se presenta en la Figura 4.23.

```
-------------------------------------------------------------------------------
-- Component R_ActuatorTorque
-------------------------------------------------------------------------------
COMPONENT R_ActuatorTorque IS_A R_Actuator
"Torque actuator"
   CONTINUOUS
      m_out.tau = s_in.signal[1]

END COMPONENT
```

Fig. 4.23.: Código del componente R_ActuatorTorque.

4.5.3.- Componentes de la subclase *BASIC*.

Una gran variedad de sistemas mecánicos se basan en movimientos de rotación alrededor de un eje fijo o no acelerado, entendiendo este movimiento de rotación como aquel en el que todas las partículas que componen el cuerpo en rotación describen trayectorias circulares en torno a este eje.

Por suerte, todos estos sistemas mecánicos, desde una simple manivela, pasando por una bomba centrífuga y hasta el mecanismo más complejo que se pueda imaginar, se pueden modelar en parte, mediante el uso de tres elementos básicos de la mecánica rotacional. Estos elementos, que integran la subclase *BASIC* de la librería *Rotational Library*, son la masa rotacional o inercia *R_Inertia*, el muelle de torsión *R_Spring* y el amortiguador de torsión *R_Damper*, y sus características más importantes se analizan a continuación.

4.5.3.1.- Componente *R_Inertia*.

Mediante este elemento se modela la inercia de cualquier cuerpo en rotación, esto es, la resistencia que este opone a modificar su estado, ya sea de reposo o movimiento. La inercia de un cuerpo (en dinámica rotacional) se mide a través de su momento de inercia.

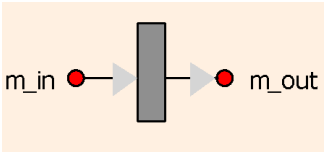
El momento de inercia es una magnitud que refleja la distribución de masa de un cuerpo o de un sistema de partículas en rotación respecto a un eje de giro. El momento de inercia sólo depende de la geometría del cuerpo y de la posición del eje de giro, pero no depende de las fuerzas que intervienen en el movimiento.

Usando la *segunda ley de Newton* se puede establecer una relación entre la inercia del sólido en rotación, la aceleración angular y el sumatorio de los momentos externos que actúan sobre sólido, descrita en la Ecuación (4.24.):

$$I \cdot \alpha = \sum tau \tag{4.24.}$$

La Ecuación (4.24.) será la que caracterice el comportamiento del componente *R_Inertia*. La descripción y la representación gráfica de este elemento se muestra en la Tabla 4.37.

Tabla 4.37.: Descripción y representación gráfica de R_Inertia.

Nombre	Descripción	Representación gráfica
R_Inertia	Define la inercia de cualquier sólido rígido en rotación	

El disco de inercia descrito en la Tabla 4.37. hereda de *R_Rigid*, que a su vez hereda de *R_two_Ports*, con lo cual *R_Inertia* adquiere las características de estas dos clases abstractas.

Los puertos que se definen dentro de *R_Inertia* son los mostrados en la Tabla 4.38.

Tabla 4.38.: Puetos para R_Inertia.

Puerto	Tipo	Dirección
m_in	*mech_rot*	IN
m_out	*mech_rot*	OUT

Las constantes y las variables introducidas en este elemento se muestran en las Tablas 4.39. y 4.40. respectivamente.

Tabla 4.39.: Constantes para R_Inertia.

Dato	Tipo	Descripción	Valor inicial	Unidades
I	REAL	Valor del momento de inercia del elemento	1	$kg \cdot m^2$
phi_0	REAL	Define la posición angular inicial del elemento	0	rad

Tabla 4.40.: Descripción variables declaradas en R_Inertia.

Variable	Tipo	Descripción	Unidades
phi	REAL	Define el posición angular absoluta del elemento	rad
alpha	REAL	Define la aceleración angular absoluta del elemento	rad/s^2

Si ahora se reescribe la Ecuación (4.24.) en función de las variables del elemento, se tiene la Ecuación (4.25.):

$$I \cdot alpha = m\_in.tau - m\_out.tau \qquad (4.25.)$$

Mediante la Ecuación (4.25.) se dota a *R_Inertia* de la capacidad de conocer el comportamiento del sólido en rotación. Pero además de esta ecuación, en el modelo se introducen las Ecuaciones (4.26.), que relaciona la aceleración angular del sólido en rotación con su posición angular, y (4.27.) que impone una condición de contorno que permite calcular todas las incógnitas del modelo:

$$phi'' = alpha \qquad (4.26.)$$

$$phi = phi\_0 \tag{4.27.}$$

El código en lenguaje de programación EL que representa el componente se muestra en la Figura 4.24.

```
-------------------------------------------------------------------------------
-- Component R_Inertia
-------------------------------------------------------------------------------
COMPONENT R_Inertia IS_A R_Rigid
"Rotating mass with inertia"
   DATA
      REAL I = 1        UNITS "kg*m^2"    "Moment of inertia of body (kg*m^2)"
      REAL phi_0 = 0    UNITS "rad"       "Initial angular position (rad)"

   DECLS
      REAL alpha        UNITS "rad/s^2"      "Absolute angular acceleration (rad/s^2)"

   INIT
      phi = phi_0

   CONTINUOUS

      phi'' = alpha
      I * alpha = m_in.tau - m_out.tau

END COMPONENT
```

Fig. 4.24.: Código del componente R_Inertia.

4.5.3.2.- Componente *R_Spring*.

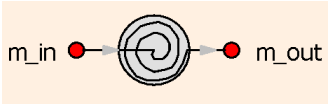
Mediante el componente *R_Spring* se consigue modelar el comportamiento de los resortes torsionales. La manera más sencilla de modelar un resorte de torsión ideal consiste en suponer que éste obedece la *Ley de Hooke*, la cual establece una relación entre el par en el resorte y la deformación angular que éste experimenta a través de una constante *k_tor*, denominada constante de rigidez, la cual se expresa en la Ecuación (4.28.):

$$tau = - k\_tor \cdot (phi\_rel - phi\_natural) \tag{4.28.}$$

La Ecuación (4.28.), que será la que caracterice el comportamiento de *R_Spring*, representa la *Ley de Hooke* para un resorte de torsión donde *tau* es el par en el resorte, *k_tor* es la constante de rigidez a torsión, *phi_rel* es la deformación que experimenta el resorte y *phi_natural* es la deformación natural del resorte. El signo negativo simboliza que el par en el resorte es opuesto al sentido de la deformación.

Además, este componente hereda todas las características de *R_Compliant*, que a su vez hereda de *R_two_Ports*, con lo que *R_Spring* adquiere las características de estos dos componentes abstractos. La descripción y la representación gráfica de *R_Spring* se muestra en la Tabla 4.41.

Tabla 4.41.: Descripción y representación gráfica de R_Spring.

Nombre	Descripción	Representación gráfica
R_Spring	Define un resorte sometido a fuerzas de torsión ideal con dos puertos mecánicos rotacionales y con movimiento relativo entre estos dos puertos	

Los puertos que posee este elemento se presentan en la Tabla 4.42.

Tabla 4.42.: Puertos para R_Spring.

Puerto	Tipo	Dirección
m_in	*mech_rot*	IN
m_out	*mech_rot*	OUT

Las constantes y variables introducidas en este elemento se muestran en las Tablas 4.43. y 4.44. respectivamente.

Tabla 4.43.: Descripción de las constantes introducidas por R_Spring.

Dato	Tipo	Descripción	Valor inicial	Unidades
k_tor	REAL	Define la constante de rigidez a torsión del resorte	1	N·m/rad
phi_natural	REAL	Define el ángulo natural entre los extremos del resorte	0	rad

Tabla 4.44.: Descripción de las variables declaradas por R_Spring.

Variable	Tipo	Descripción	Unidades
phi_rel	REAL	Define el ángulo de giro relativo	rad

Mediante la Ecuación (4.28.) se puede escribir el código en lenguaje de programación EL que representa al componente *R_Spring* como en la Figura 4.25.

```
-------------------------------------------------------------------------------
-- Component R_Spring
-------------------------------------------------------------------------------
COMPONENT R_Spring IS_A R_Compliant
"Ideal torsional spring"
   DATA
      REAL k_tor = 1           UNITS "N*m/rad"    "Spring constant (N*m/rad)"
      REAL phi_natural = 0     UNITS "rad"        "Angular distance between ports for unstretched spring (rad)"

   CONTINUOUS
      tau = - k_tor * (phi_rel - phi_natural)

END COMPONENT
```

Fig. 4.25.: Código del componente R_Spring en EL.

4.5.3.3.- Componente *R_Damper*.

El amortiguador es un dispositivo que absorbe energía, utilizado normalmente para disminuir las oscilaciones no deseadas de un movimiento periódico o para absorber energía proveniente de golpes o impactos.

Existen diversas formas de modelar el amortiguamiento, pero la más simple de todas ellas consta de una partícula o masa concentrada, que va perdiendo velocidad bajo la acción de una fuerza de amortiguamiento proporcional a su velocidad, descrita en la Ecuación (4.29.):

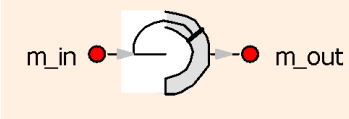
$$F = -c \cdot \frac{dx}{dt} \tag{4.29.}$$

La Ecuación (4.29.) representa la fuerza de amortiguamiento para un movimiento de traslación. Si se lleva dicha ecuación al movimiento rotacional se obtiene la Ecuación (4.30.), que caracterizará el comportamiento del componente *R_Damper*:

$$tau = -c_{tor} \cdot w_{rel} \tag{4.30.}$$

En la Ecuación (4.30.), *tau* representa el par que se opone al movimiento, *c_tor* es el amortiguamiento del sistema y *w_rel* es la velocidad relativa entre los extremos del amortiguador. La descripción de R_Damper y su representación gráfica se muestra en la Tabla 4.45.

Tabla 4.45.: Descripción y representación gráfica de R_Damper.

Nombre	Descripción	Representación gráfica
R_Damper	Define un amortiguador rotacional ideal con dos puertos mecánicos rotacionales	

Los puertos que posee este elemento se presentan en la Tabla 4.46.

Tabla 4.46.: Puertos para R_Damper.

Puerto	Tipo	Dirección
m_in	*mech_rot*	IN
m_out	*mech_rot*	OUT

La constante y la variable introducidas en este elemento se muestran en las Tablas 4.47. y 4.48. respectivamente.

Tabla 4.47.: Descripción de la constante introducidas por R_Damper.

Dato	Tipo	Descripción	Valor inicial	Unidades
c_tor	REAL	Define la constante de amortiguamiento de *R_Damper*	1	N·m·s/rad

Tabla 4.48.: Descripción de la variable declaradas por R_Damper.

Variable	Tipo	Descripción	Unidades
w_rel	REAL	Define la velocidad angular relativa entre los extremos del componente	rad/s

El código en lenguaje de programación EL que representa el componente *R_Damper*, donde hay que modelar la Ecuación (4.30.), se muestra en la Figura 4.26.

```
-------------------------------------------------------------------------------
-- Component R_Damper
-------------------------------------------------------------------------------
COMPONENT R_Damper IS_A R_Compliant
"Ideal rotational damper"
   DATA
      REAL c_tor = 1   UNITS "N*m*s/rad"    "Damping constant (N*m*s/rad)"

   DECLS
      REAL w_rel       UNITS "rad/s"        "Relative angular velocity between ports (rad/s)"

   CONTINUOUS

      w_rel = m_out.phi' - m_in.phi'

      tau = - c_tor * w_rel

END COMPONENT
```

Fig. 4.26.: Código del componente R_Damper en EL.

4.5.4.- Componentes de la subclase *BEARING*.

En las máquinas es frecuente encontrar interacciones entre piezas con deslizamiento relativo en las que es necesario reducir la fricción y minimizar el desgaste. Esto se consigue mediante unos elementos denominados cojinetes. Dentro de la subclase *BEARING* se va a incluir el componente *R_Hydro_Bearing*, cojinete con lubricación hidrodinámica.

4.5.4.1.- Componente *R_Hydro_Bearing*.

Un cojinete es un dispositivo que permite el movimiento relativo entre superficies, minimizando la pérdida de energía y el desgaste de las mismas. Existen diferentes disposiciones constructivas y múltiples regímenes de lubricación para satisfacer los requerimientos cinemáticos y de carga de un cojinete, de entre todas ellas, el componente *R_Hydro_Bearing* representará un cojinete cilíndrico, como el de la Figura 4.27., con lubricación hidrodinámica.

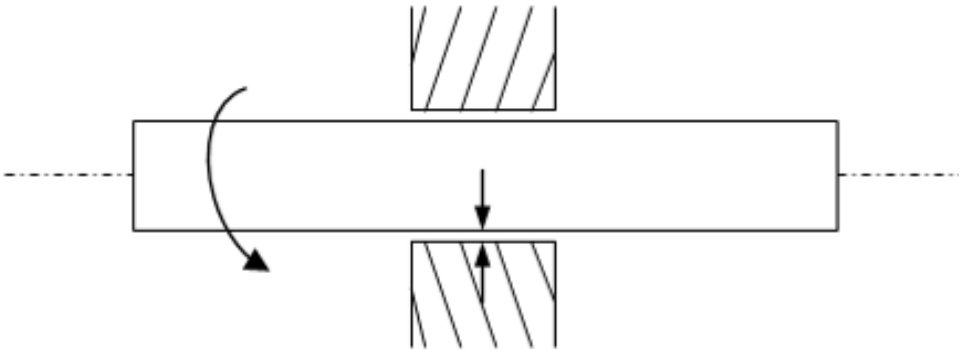


Fig. 4.27.: Esquema básico de cojinete cilíndrico.

La lubricación consiste en interponer una película de lubricante entre dos superficies con movimiento relativo, de forma que las pérdidas de energía no sean consecuencia del rozamiento entre superficies sólidas, sino de efectos de viscosidad en el fluido lubricante debido a tensiones de cortadura. En el caso de lubricación hidrodinámica, la película de lubricante es presurizada mediante efectos hidrodinámicos producidos por la combinación de una disposición geométrica y un movimiento relativo entre superficies adecuados y por el carácter viscoso del fluido.

Los orígenes de la teoría de lubricación hidrodinámica se remontan a finales del siglo XIX, como consecuencia de los estudios del investigador Beauchamp Tower acerca de la lubricación en los cojinetes de ejes de ferrocarriles. Tower abrió un orificio en la parte superior de unos de los cojinetes que ensayaba y descubrió que las presiones que el lubricante alcanzaba dentro del mismo eran muy elevadas [SAN12].

Los resultados obtenidos por Tower, llevaron a Osborne Reynolds a pensar que debía existir una ley que relacionara la presión del fluido, la velocidad relativa y el coeficiente de fricción, obteniendo una ecuación diferencial que, aunque no tiene una solución general, sigue siendo el punto de partida para los actuales estudios de lubricación [LOP12].

Sin embargo, A.A. Raimondi y John Boyd, de los *Whestinhouse Research Laboratories*, emplearon técnicas de iteración por ordenador para resolver la ecuación de Reynolds. Los resultados fueron representados en forma de gráficos, como el de la Figura 4.28., que utilizan en el eje de abcisas el número de Sommerfeld o número característico del cojinete dado por la Ecuación (4.31.) [SAN12]:

$$S = \left(\frac{r}{c}\right)^2 \frac{\eta N}{P} \tag{4.31.}$$

En la Ecuación (4.31.), r es el radio del muñón, c la holgura radial, η la viscosidad absoluta, N la velocidad relativa entre el muñón y el cojinete, y P la carga por unidad de área proyectada.

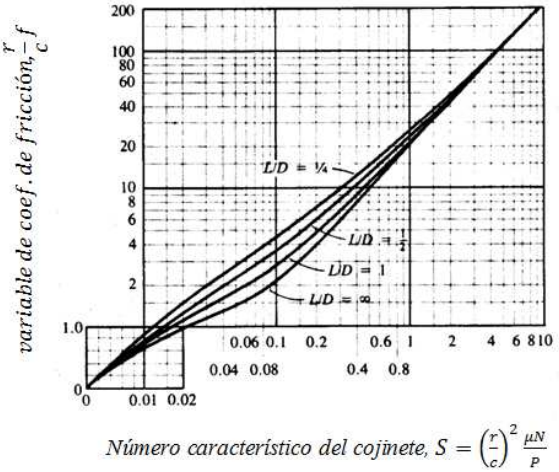
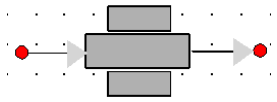


Fig. 4.28.: Diagrama para determinar la variable de coeficiente de fricción.

La descripción y la representación gráfica de *R_Hydro_Bearing* se muestran en la Tabla 4.49.

Tabla 4.49.: Descripción y representación gráfica de R_Hydro_Bearing.

Nombre	Descripción	Representación gráfica
R_Hydro_Bearing	Define un cojinete cilíndrico con lubricación hidrodinámica	

Puesto que *R_Hydro_Bearing* hereda las características de *R_Rigid*, está formado por dos puertos de tipo *mech_rot*, que se describen en la Tabla 4.50.

Tabla 4.50.: Descripción de los puertos de R_Hydro_Bearing.

Puerto	Tipo	Dirección
m_in	*mech_rot*	IN
m_out	*mech_rot*	OUT

Las constantes y variables que se introducen en este elemento se muestran en las Tablas 4.51. y 4.52. respectivamente.

Tabla 4.51.: Descripción de las constantes introducidas por R_Hydro_Bearing.

Dato	Tipo	Descripción	Valor inicial	Unidades
c	REAL	Define la holgura radial	-	mm
v	REAL	Define la viscosidad dinámica del lubricante	-	Pa·s
l	REAL	Define la longitud del cojinete	-	mm
d	REAL	Define el diámetro del muñón del cojinete	-	mm
W	REAL	Define la carga que soporta el cojinete	-	N

Tabla 4.52.: Descripción de las variables que introduce R_Hydro_Bearing.

Variable	Tipo	Descripción	Unidades
P	REAL	Define la carga que soporta el cojinete por unidad de área proyectada	N/m^2
S	REAL	Define el número de Sommerfeld	-
H	REAL	Define la potencia perdida en el cojinete	W
f	REAL	Define el coeficiente de fricción equivalente (estos datos se hallaran mediante tablas)	-
Table1D	REAL	Tabla que incluye los valores del término $\frac{r}{c}f$ en función del número de Sommerfeld y de la relación de aspecto l/d	

Dentro del componente *R_Hydro_Bearing* aparece el fragmento de código de la Figura 4.29., mediante el cual se asignan los valores a la variable *Table1D* en función de la relación de aspecto $\frac{l}{d}$ seleccionada.

```
--Asign values to Table1d depending on the relationship l/d
INIT
   IF(l/d == 1) THEN
      readTable1D("f_1.txt",table1, 1)
   ELSEIF(l/d == 1/4) THEN
      readTable1D("f_1-4.txt",table1, 1)
   ELSEIF(l/d == 1/2) THEN
      readTable1D("f_1-2.txt",table1, 1)
   ELSEIF(l/d == 1) THEN
      readTable1D("f_inf.txt",table1, 1)
   END IF
```

Fig. 4.29.: Inicialización de la variable Table1D.

Así, si la relación de aspecto equivale a la unidad, se asigna a *Table1D* los valores de la tabla *f_1.txt;* para relaciones de aspecto de 1/4, se asigna a *Table1D* los valores de la tabla *f_1-4.txt;* para relaciones de aspecto de 1/2, se asigna a *Table1D* los valores de la tabla *f_1-2.txt;* y por último, cuando la longitud es mucho mayor que el diámetro, se asigna a *Table1D* los valores de la tabla *f_inf.txt*. De esta forma, se consigue que el componente sea capaz de leer el diagrama de la Figura 4.28., en el que se obtiene el valor de la variable de fricción a partir del número de Sommerfeld y la relación de aspecto del cojinete.

Los valores de *f_1.txt, f_1-4.txt, f_1-2.txt y f_inf.txt*, se presentan en la Tabla 4.53.

Tabla 4.53.: Valores para la variable de fricción; a) l/d = 1 ; b) l/d = 1/4; c)l/d=1/2; d)l/d=∞.

a)

l/d = 1	
S	$\frac{r}{c}f$
0.000	0.000
0.002	0.250
0.004	0.450
0.006	0.600
0.008	0.750
0.010	0.900
0.012	1.000
0.020	1.400
0.030	1.800
0.040	2.150
0.045	2.800
0.060	3.200
0.080	4.250
0.100	4.500
0.200	7.250
0.400	12.000
0.600	17.500
0.800	20.000
1.000	24.000
1.200	28.000
1.400	32.000
1.600	38.000
2.000	45.000
2.400	57.500
4.000	90.000
6.000	120.000
8.000	160.000
10.000	200.000

b)

l/d = 1/4	
S	$\frac{r}{c}f$
0.000	0.000
0.002	0.250
0.004	0.450
0.006	0.550
0.008	0.650
0.010	0.760
0.012	0.850
0.020	1.160
0.030	1.380
0.040	1.580
0.044	1.670
0.048	1.780
0.052	1.900
0.056	1.990
0.060	2.010
0.080	2.400
0.100	2.800
0.120	3.100
0.140	3.500
0.160	3.900
0.200	4.600
0.240	5.400
0.300	6.500
0.400	8.250
0.488	10.000
0.600	12.000
0.700	14.000
0.800	16.000
1.000	20.000
1.400	28.000
1.600	32.000
2.000	40.000
10.000	200.000

Tabla 4.53. (continuación): Valores para la variable de fricción; a) l/d = 1 ; b) l/d = 1/4; c)l/d=1/2; d)l/d=∞.

c)

l/d = 1/2	
S	$\frac{r}{c}f$
0.000	0.000
0.002	0.250
0.004	0.450
0.006	0.600
0.009	0.800
0.010	0.800
0.012	0.950
0.014	1.000
0.016	1.150
0.020	1.300
0.030	1.600
0.040	1.900
0.050	2.190
0.060	2.400
0.080	3.000
0.100	3.600
0.200	5.200
0.300	7.500
0.400	9.500
0.600	14.000
0.800	17.500
1.000	22.000
1.200	26.000
1.400	32.000
1.600	38.000
2.000	45.000
2.400	57.500
4.000	90.000
6.000	120.000
8.000	160.000
10.000	200.000

d)

l/d = ∞	
S	$\frac{r}{c}f$
0.000	0.000
0.002	0.150
0.004	0.300
0.006	0.450
0.008	0.550
0.010	0.650
0.012	0.720
0.014	0.800
0.016	0.825
0.020	0.900
0.030	1.100
0.040	1.210
0.050	1.390
0.060	1.480
0.080	1.750
0.100	2.000
0.120	2.350
0.140	2.700
0.180	3.420
0.200	3.800
0.220	4.300
0.240	4.500
0.280	5.250
0.300	5.500
0.320	5.900
0.340	6.400
0.380	7.100
0.400	7.500
0.450	8.500
0.500	9.500
0.550	10.000
0.600	14.000
0.800	17.500
1.000	22.000
1.200	26.000
1.400	32.000
1.600	38.000
2.000	45.000
2.400	57.500
4.000	90.000
6.000	120.000
8.000	160.000
10.000	200.000

Una vez que el componente *R_Hydro_Bearing* puede interpretar qué curva de la Figura 4.28. tiene que utilizar, es posible determinar el valor del coeficiente de fricción *f*. Para ello, se realiza una interpolación lineal en la tabla *Tabla_1D* para el valor del número característico del cojinete *S* obtenido según la Ecuación (4.31.), donde *P* se calcula como según la Ecuación (4.32.):

$$P = \frac{W}{l \cdot d} \tag{4.32.}$$

El resultado obtenido en la interpolación permite obtener el valor del coeficiente de fricción para el cojinete simulado de la Ecuación (4.33.):

$$f = \frac{r}{c} \cdot Interpolación(Table_{1D}, S) \tag{4.33.}$$

Una vez conocido el valor de *f*, la potencia perdida en el cojinete se calcula mediante la Ecuación (4.34.), donde *r* representa el radio del cojinete, *W* la carga que soporta el cojinete, *phi'* la velocidad angular del cojinete y *f* representa el coeficiente de rozamiento equivalente:

$$H = f \cdot W \cdot phi' \cdot r \tag{4.34.}$$

A partir de la pérdida de potencia *H*, y mediante un equilibrio de momentos en el cojinete, se determina el par a la salida del cojinete.

El código en lenguaje de programación EL que representa el componente *R_Hydro_Bearing* se muestra en la Figura 4.30.

```
-------------------------------------------------------------------------------
-- Component R_Hydro_Bearing
-------------------------------------------------------------------------------
COMPONENT R_Hydro_Bearing IS_A R_Rigid
"Cylindrical bearing with hydrodinamic lubrication"
   DATA
      REAL c  UNITS   "mm"       "Radial play (mm)"
      REAL v  UNITS   "Pa*s"     "Lubricant viscosity (Pa*s)"
      REAL l  UNITS   "mm"       "Bearing length (mm)"
      REAL d  UNITS   "mm"       "Bearing diameter (mm)"
      REAL W  UNITS   "N"        "Bearing load (N)"

   DECLS
      REAL S  UNITS   "-"        "Sommerfeld number (-)"
      REAL P  UNITS   "N/m^2"    "Bearing load on projected area (N/m^2)"
      REAL H  UNITS   "W"        "Power loss (W)"
      REAL f  UNITS   "-"        "Equivalent friction coefficient (-)"
      TABLE_1D table1

   --Asign values to Table1d depending on the relationship l/d
   INIT
      IF(l/d == 1) THEN
         readTable1D("f_1.txt",table1, 1)
      ELSEIF(l/d == 1/4) THEN
         readTable1D("f_1-4.txt",table1, 1)
      ELSEIF(l/d == 1/2) THEN
         readTable1D("f_1-2.txt",table1, 1)
      ELSEIF(l/d == 1) THEN
         readTable1D("f_inf.txt",table1, 1)
      END IF

   CONTINUOUS
      --Load on projected area
      P=(W/(l*d*(10**-6)))

      --Sommerfeld number
      S=((d/(2*c))**2)*((v*m_in.phi')/(2*P*MATH_ELEMENTS.PI))

      -- Friction coef. depending on Sommerfeld number
      f=linearInterp1D(table1,S)*((2*c)/d)

      --Power losses
      H=f*W*m_in.phi'*(d/2000)

      --Torque balance for the bearing
      m_out.tau+(H/m_in.phi')=m_in.tau

END COMPONENT
```

Fig. 4.30.: Código del componente R_Hydro_Bearing en EL.

4.5.5.- Componentes de la subclase *GEARS*.

En la subclase *GEARS* de la librería *Rotational Library* se van a incluir todos los elementos donde intervienen ruedas dentadas que engranan entre sí, desde simples engranajes sin fricción, hasta cajas de cambios con fricción hidrodinámica. Además, se incluirán los componentes usados para modelar de una manera más real los engranajes, como es el caso de un componente que simula la eficiencia y un componente que simula la holgura o juego entre dientes de una pareja de engranajes.

4.5.5.1.- Componente *R_GearIdeal*.

El componente *R_GearIdeal*, pretende modelar el comportamiento de un engranaje, entendido éste como aquel elemento que se utiliza para transmitir potencia de un eje a otro de una máquina. Por tanto, estos elementos permiten modificar la velocidad de giro y el par transmitido en virtud de un parámetro denominado relación de transmisión, que se obtiene a partir del cociente entre la velocidad de entrada y salida del engranaje. En la Ecuación (4.35.) se tiene la relación de transmisión de un engranaje:

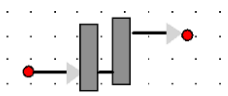
$$i = \frac{w_{entrada}}{w_{salida}}\,[-] \tag{4.35.}$$

Además, y dado que la potencia a un lado y a otro del engranaje debe permanecer constante, se tiene la relación entre los pares de entrada y salida de la Ecuación (4.36.):

$$T_{salida} = i \cdot T_{entrada} \tag{4.36.}$$

La descripción y la representación gráfica de *R_GearIdeal* se muestra en la Tabla 4.54.

Tabla 4.54.: Descripción y representación gráfica de R_GearIdeal.

Nombre	Descripción	Representación gráfica
R_GearIdeal	Define un engranaje con una relación de transmisión dada	

El componente *R_GearIdeal* hereda de *R_Two_Ports*, por tanto tendrá dos puertos tipo *mech_rot*, los cuales se describen en la Tabla 4.55.

Tabla 4.55.: Descripción de los puertos de R_GearIdeal.

Puerto	Tipo	Dirección
m_in	*mech_rot*	IN
m_out	*mech_rot*	OUT

La única constante que se introduce en este componente es la relación de transmisión mostrada en la Tabla 4.56.

Tabla 4.56.: Constante introducida por R_GearIdeal.

Dato	Tipo	Descripción	Valor inicial	Unidades
ratio	REAL	Define la relación de transmisión	1	mm

El código en EL del componente *R_GearIdeal* se define en la Figura 4.31.

```
-----------------------------------------------------------------
-- Component R_GearIdeal
-----------------------------------------------------------------
COMPONENT R_GearIdeal IS_A R_Two_Ports
"Ideal Gear without inertia"
   DATA
      REAL ratio = 1    UNITS "-"   "Transmission ratio (-)"

   CONTINUOUS
      m_in.phi' = ratio * m_out.phi'
      ratio * m_in.tau = m_out.tau

END COMPONENT
```

Fig. 4.31.: Código del componente R_GearIdeal en EL.

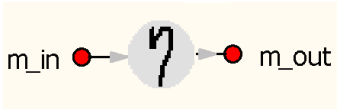
4.5.5.2.- Componente *R_Efficiency.*

La eficiencia o rendimiento de un dispositivo es la relación entre la potencia útil y la potencia invertida. La potencia real de salida en un engranaje, o de una caja de cambios, es la potencia de entrada menos las pérdidas de energía asociadas principalmente por la fricción y por la lubricación entre los dientes de los engranajes. Estas pérdidas son relativamente independientes de la naturaleza de los engranajes y de las relaciones de transmisión. Calcular estas pérdidas con exactitud resulta muy complicado y hay que hacer estimaciones basadas en la experiencia. Las pérdidas por fricción están relacionadas con el diseño de engranajes, el ángulo de presión, el tamaño del engranaje, y el coeficiente de fricción [BEA12].

Para poder tratar una caja de cambios de manera real, o cualquier otro dispositivo mecánico con pérdidas, es necesario tener en cuenta el rendimiento o eficiencia de la misma. Este valor se puede obtener de manera experimental o puede ser dado directamente como dato. En este proyecto final de carrera, para simular sistemas donde intervengan cajas de cambios, se va a suponer un valor de coeficiente de eficiencia *eta* dado, pero también se va a explicar cómo se podría hallar con una simulación de forma matemática en el Apartado 5.3., donde se calculará el valor de la eficiencia de forma numérica y se contrastará con el valor hallado de forma experimental en EcosimPro.

A continuación se va a crear el modelo del componente en lenguaje EL que representa la influencia del rendimiento en un elemento mecánico rotacional, al que se le llamará *R_Efficiency*. Este componente va a tener dos puertos, uno de entrada y otro de salida, y va a heredar de la clase abstracta *R_Rigid*. En la Tabla 4.57. se muestra una breve descripción y la representación gráfica del componente.

Tabla 4.57.: Descripción y representación gráfica de R_Efficiency.

Nombre	Descripción	Representación gráfica
R_Efficiency	Define la eficiencia o rendimiento de un elemento mecánico	

Los puertos que posee este elemento se presentan en la Tabla 4.58.

Tabla 4.58.: Puertos para R_Efficiency.

Puerto	Tipo	Dirección
m_in	*mech_rot*	IN
m_out	*mech_rot*	OUT

Las constantes y variables introducidas en este elemento se muestran en las Tablas 4.59. y 4.60. respectivamente.

Tabla 4.59.: Descripción de las constantes introducidas por R_Efficiency.

Dato	Tipo	Descripción	Valor inicial	Unidades
eta	REAL	Define la eficiencia o rendimiento del elemento mecánico	1	-

Tabla 4.60.: Descripción de las variables declaradas por R_Efficiency.

Variable	Tipo	Descripción	Unidades
P_r	REAL	Define la potencia introducida al elemento mecánico	W

El valor de la eficiencia *eta* está acotado y sólo puede tomar valores entre 0 y 1. La potencia en un elemento mecánico rotacional se calcula según la Ecuación (4.37.) donde se multiplica el valor del par de entrada $tau_{entrada}$ por el valor de la velocidad angular ω, es importante indicar que para que este modelo sea válido la velocidad angular de entrada debe ser igual a la velocidad angular de salida:

$$P = tau_{entrada} \cdot \omega \tag{4.37.}$$

Para calcular el valor del par de salida tau_{salida} hay que pensar en dos posibles situaciones diferentes. Por una parte, puede ocurrir que el valor de la potencia de entrada sea mayor o igual que 0, con lo que para calcular el valor del par de salida, o par real, habrá que multiplicar el valor del par de entrada $tau_{entrada}$ por la eficiencia *eta*, este caso se representa con la Ecuación (4.38.). Mientras que si el valor de entrada es menor que 0, habrá que dividir el valor del par de entrada entre el valor de la eficiencia *eta*, condición de la Ecuación (4.39.):

$$tau_{salida} = tau_{entrada} \cdot tau \tag{4.38.}$$

$$tau_{salida} = \frac{tau_{entrada}}{tau} \tag{4.39.}$$

El código en lenguaje de programación EL que representa al componente *R_Efficiency* viene dado en la Figura 4.32.

```
-------------------------------------------------------------------------------
-- Component R_Efficiency
-------------------------------------------------------------------------------
COMPONENT R_Efficiency IS_A R_Rigid
"Mechanical efficiency"
   DATA
      REAL eta = 1     UNITS "-"      "Efficiency: [0,1] (-)"

   DECLS
      REAL P_r         UNITS "W"      "Power (W)"

   CONTINUOUS
      P_r = m_in.tau * phi'

      m_out.tau = ZONE (P_r >= 0) eta * m_in.tau
                  OTHERS          m_in.tau / eta

END COMPONENT
```

Fig. 4.32.: Código del componente R_Efficiency en EL.

4.5.5.3.- Componente *R_ElastoBacklash.*

El *backlash* en las transmisiones por engranajes es el huelgo o juego que se deja entre los dientes que engranan. Normalmente es necesario un cierto grado de huelgo para prevenir que los dientes se traben entre sí cuando aumenta la temperatura. Un huelgo excesivo se considera un error de montaje ya que está directamente relacionado con la distancia que existe entre los ejes de los engranajes. Cuando hay un huelgo excesivo en las trasmisiones por engranajes aparecen vibraciones importantes que pueden provocar que los dientes no engranen exactamente donde deben, por lo que un diente podría engranar después de que los dientes previos dejen de hacer contacto, produciéndose una aceleración en el sistema que provocará que cuando el diente logre hacer contacto, lo haga de forma brusca pudiendo rebotar y hacer contacto con su perfil posterior en el diente que esta tras él. Cuanto más cerca se encuentren los ejes de rotación menor será el huelgo [THE00].

Un par de engranajes podría diseñarse para tener holgura cero, pero esto supondría un perfecto diseño de los engranajes, una no lubricación entre los engranajes y que las características de expansión térmicas sean uniformes en todo el sistema. Esto aumentaría en exceso el coste del sistema mecánico, con lo cual, las parejas de engranajes se suelen diseñar de manera que exista un pequeño huelgo entre sus dientes. En la Figura 4.33. aparecen dos imágenes de parejas de engranajes donde existe huelgo o *backlash* reducido (Figura 4.33.a) y huelgo excesivo (Figura 4.33.b).

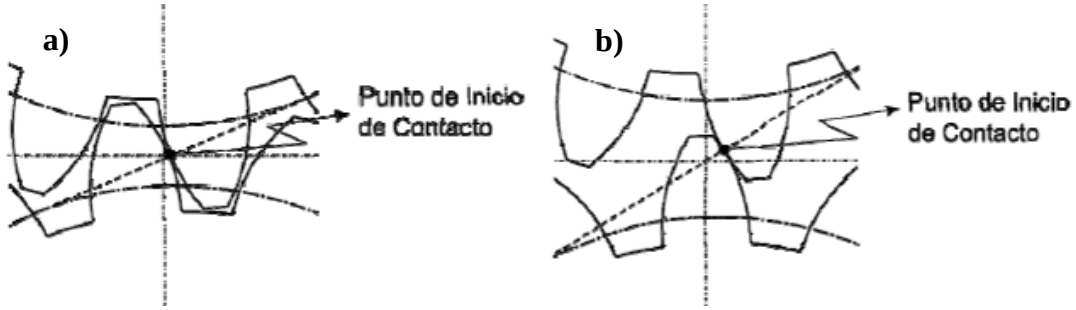


Fig. 4.33.: Engrane para distintas posiciones: a) en un punto alto, poco huelgo; b) en un punto bajo, excesivo huelgo.

Se puede ver en la Figura 4.33. como el punto de inicio de contacto es distinto para ambos casos a pesar de que se trata de la misma pareja de engranajes [DED99].

Para situaciones en las que la precisión es importante, el contragolpe provocado por el huelgo o *backlash* puede ser minimizado a través de varias técnicas llamadas anti-huelgo. Una de estas técnicas para reducir el huelgo consiste en estrechar los dientes en la dirección axial, de modo que deslizando los engranajes en esta dirección el juego entre dientes se reduce. Pero la técnica más habitual, y por eso se va a crear el modelo *R_ElstoBacklash* en la librería *Rotational Library* en base a ella, es dividir el engranaje a lo largo de un plano perpendicular al eje de manera que una mitad quede fija al eje de forma normal y la otra mitad quede libre girando alrededor del eje, añadiendo resortes que proporcionen un par relativo entre las dos mitades, produciéndose así una expansión entre los dientes del engranaje. En la Figura 4.34. se aprecia una imagen en la que aparece un engranaje con resortes que minimizan el huelgo entre dientes para sistemas de engranajes.

Fig. 4.34.: Engranaje con sistema anti-huelgo.

El elemento *R_ElastoBacklash* va a consistir en un mecanismo anti-huelgo conectado en serie con un resorte y un amortiguador, ambos rotacionales, conectados a su vez en paralelo. La forma de modelar este resorte y este amortiguador rotacional va a ser la misma que la utilizada en los elementos *R_Spring* y *R_Damper* de la Sección 4.5.3., donde la constante del resorte no puede ser cero, de lo contrario, este componente no podría ser usado. Para modelar este componente se van a usar las Ecuaciones (4.28.) y (4.30.), vistas con anterioridad. Por otro lado, el valor del par resultante *tau* dependerá de si el ángulo de giro relativo *phi_rel* es mayor o igual que el ángulo muerto directo *ddz*, o menor o igual que el ángulo muerto inverso *idz*. Si estamos fuera de estos rangos, el par resultante *tau* será nulo. Estas restricciones se pueden expresar programando el bloque de la Figura 4.35. en el modelo del componente.

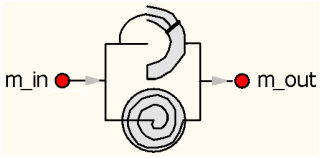
```
tau = ZONE (phi_rel >= ddz) - k_tor * (phi_rel - phi_natural - ddz) - c_tor * w_rel
      ZONE (phi_rel <= idz) - k_tor * (phi_rel - phi_natural + idz) - c_tor * w_rel
      OTHERS                          0
```

Fig. 4.35.: Código del componente R_ElastoBacklash en EL.

Nótese que los valores del par resultante *tau* serán la suma del par resultante dado por el amortiguador más el par resultante dado por el resorte, pero restando y sumando en la deformación angular de este último el valor del ángulo muerto, dependiendo de la zona

que corresponda. Además, este componente hereda todas las características de *R_Compliant*, que a su vez hereda de *R_two_Ports*, con lo que *R_ElastoBacklash* adquiere las características de estas dos clases abstractas. La descripción y la representación gráfica de *R_ElastoBacklash* se muestran en la Tabla 4.61.

Tabla 4.61.: Descripción y representación gráfica de R_ElastoBacklash.

Nombre	Descripción	Representación gráfica
R_ElastoBacklash	Define un sistema anti-huelgo con elasticidad y amortiguación	

Los puertos que posee este elemento se presentan en la Tabla 4.62.

Tabla 4.62.: Puertos para R_ElastoBacklash.

Puerto	Tipo	Dirección
m_in	*mech_rot*	IN
m_out	*mech_rot*	OUT

Las constantes y variables introducidas en este elemento se muestran en las Tablas 4.63. y 4.64. respectivamente.

Tabla 4.63.: Descripción de las constantes introducidas por R_ ElastoBacklash.

Dato	Tipo	Descripción	Valor inicial	Unidades
k_tor	REAL	Define la constante de rigidez a torsión del resorte	1	N·m/rad
phi_natural	REAL	Define el ángulo natural entre los extremos del resorte	0	rad
c_tor	REAL	Define la constante de amortiguamiento de *R_Damper*	1	N·m·s/rad
ddz	REAL	Define el ángulo de zona muerta directa	0	rad
idz	REAL	Define el ángulo de zona muerta inversa	0	rad

Tabla 4.64.: Descripción de las variables declaradas por R_ ElastoBacklash.

Variable	Tipo	Descripción	Unidades
phi_rel	REAL	Define el ángulo de giro relativo	rad
w_rel	REAL	Define la velocidad angular relativa entre los extremos del componente	rad/s

El código en lenguaje de programación EL que representa el componente *R_ElastoBacklash* se muestra en la Figura 4.36.

```
-------------------------------------------------------------------------------
-- Component R_ElastoBacklash
-------------------------------------------------------------------------------
COMPONENT R_ElastoBacklash IS_A R_Compliant
"Helicoidal spring and damper with backlash"
   DATA
      REAL c_tor = 1          UNITS "N*m*s/rad"   "Damping constant (N*m*s/rad)"
      REAL k_tor = 1          UNITS "N*m/rad"     "Spring constant (N*m/rad)"
      REAL phi_natural = 0    UNITS "rad"         "Angular distance between ports for unstretched spring (rad)"
      REAL ddz = 0            UNITS "rad"         "Direct dead zone (rad)"
      REAL idz = 0            UNITS "rad"         "Inverse dead zone (rad)"

   DECLS
      REAL w_rel              UNITS "rad/s"       "Relative angular velocity between ports (rad/s)"

   CONTINUOUS
      w_rel = m_out.phi' - m_in.phi'

      tau = ZONE (phi_rel >= ddz) - k_tor * (phi_rel - phi_natural - ddz) - c_tor * w_rel
            ZONE (phi_rel <= idz) - k_tor * (phi_rel - phi_natural + idz) - c_tor * w_rel
            OTHERS                          0

END COMPONENT
```

Fig. 4.36.: Código del componente R_ElastoBacklash en EL.

4.5.5.4.- Componente *R_Gear_SNH*.

Este componente modelará los efectos de una caja de cambios, en particular será una caja de cambios con un rendimiento dado, debido a la fricción que hay entre los dientes de los engranajes, con cojinetes hidrodinámicos, y con un sistema anti-huelgo dado por un resorte rotacional y un amortiguador rotacional. Para poder simular el sistema descrito, es necesario unir varios componentes modelados anteriormente en la librería *Rotational Library*, que son: *R_GearIdeal*, *R_Hydro_Bearing*, *R_Efficiency* y *R_ElastoBacklash*.

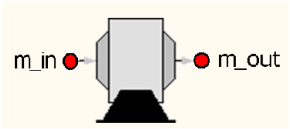
En este proyecto final de carrera, se ha decidido usar un modelo de lubricación hidrodinámica para engranajes, ya que en algún momento de velocidades críticas la lubricación limítrofe desaparece y da lugar a una lubricación hidrodinámica. Esto sucede cuando las superficies de los engranajes están completamente cubiertas con una película de lubricante como se puede apreciar en la Figura 4.37.

Fig. 4.37.: Lubricación hidrodinámica en engranajes.

Esta condición se da una vez que una película de lubricante se mantiene entre los engranajes y la presión del lubricante crea una ola de lubricante delante de la película que impide el contacto entre superficies. Bajo condiciones hidrodinámicas, no hay contacto físico entre los componentes y no hay desgaste. Si las cajas de cambios pudieran funcionar bajo condiciones hidrodinámicas todo el tiempo, no habría necesidad de utilizar ingredientes anti-desgaste y de alta presión en las fórmulas de lubricantes, el desgaste sería mínimo, pero esto es imposible de conseguir [PAL06].

La descripción y la representación gráfica de *R_Gear_SNH* se muestra en la Tabla 4.65.

Tabla 4.65.: Descripción y representación gráfica de R_Gear_SNH.

Nombre	Descripción	Representación gráfica
R_Gear_SNH	Define una caja de cambios con: un rendimiento dado, cojinetes hidrodinámicos y un sistema anti-huelgo generado por un resorte rotacional y un amortiguador rotacional	

El componente *R_Gear_SNH* hereda de *R_Two_Ports*, por tanto tendrá dos puertos tipo *mech_rot* , los cuales se describen en la Tabla 4.66.

Tabla 4.66.: Descripción de los puertos de R_Gear_SNH.

Puerto	Tipo	Dirección
m_in	*mech_rot*	IN
m_out	*mech_rot*	OUT

Las constantes introducidas en este elemento son las mismas que las introducidas en los elementos: *R_GearIdeal*, *R_Hydro_Bearing*, *R_Efficiency* y *R_ElastoBacklash*, se muestran en la Tabla 4.67.

Tabla 4.67.: Descripción de las constantes introducidas por R_ Gear_SNH.

Dato	Tipo	Descripción	Valor inicial	Unidades
ratio	REAL	Define la relación de transmisión	1	mm
eta	REAL	Define la eficiencia o rendimiento del elemento mecánico	1	-
c	REAL	Define la holgura radial	-	mm
v	REAL	Define la viscosidad dinámica del lubricante	-	Pa·s
l	REAL	Define la longitud del cojinete	-	mm
d	REAL	Define el diámetro del muñón del cojinete	-	mm
W	REAL	Define la carga que soporta el cojinete	-	N
k_tor	REAL	Define la constante de rigidez a torsión del resorte	1	N·m/rad
c_tor	REAL	Define la constante de amortiguamiento de *R_Damper*	1	N·m·s/rad
b	REAL	Define el ángulo de zona muerta	0	rad
phi_rel_0	REAL	Define la distancia angular inicial entre los dos puertos	0	rad

Para simular este componente se puede tratar como un sistema, ya que como se ha dicho, está compuesto por varios componentes simulados anteriormente en el presente proyecto. Al ser un conjunto de varios componentes, todas las ecuaciones que describen el modelo de dicho componente ya se han presentado en este apartado y en el apartado anterior, el cual hablaba de cojinetes, son las Ecuaciones 4.28., 4.30., 4.31., 4.32., 4.33., 4.34., 4.35., 4.36., 4.37., 4.38. y 4.39. Antes de presentar el código del componente modelado en la pantalla *"Code View"*, se va a mostrar una imagen en la Figura 4.38. donde se puede ver que es posible llegar a este mismo código mediante la pantalla *"Schematic view"*, simplemente uniendo los componentes que forman el sistema en el orden deseado y añadiendo dos puertos mecánicos rotacionales, *mech_rot*:

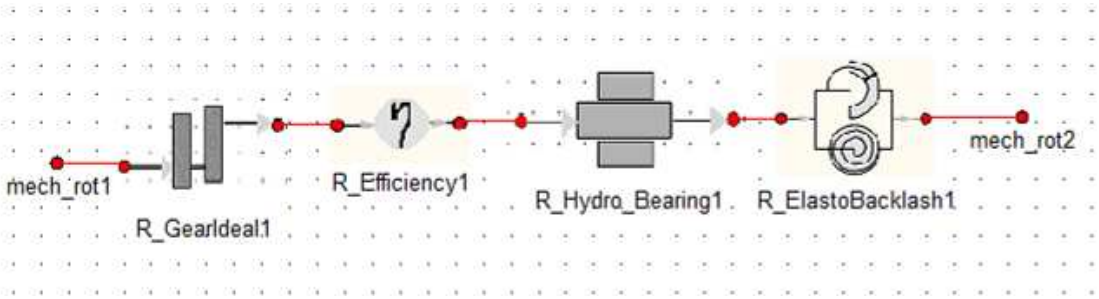


Fig. 4.38.:Vista en la pantalla "Schematic view" del sistema R_Gear_SNH.

El código en lenguaje de programación EL que representa el componente *R_Gear_SNH* se muestra en la Figura 4.39.

```
-------------------------------------------------------------------------------
-- Component R_Gear_SNH
-------------------------------------------------------------------------------
COMPONENT R_Gear IS_A R_Two_Ports
"Gearbox with hydrodinamic lubrication, efficiency and elastobacklash"
   DATA
      REAL ratio = 1       UNITS "-"           "Transmision ratio (-)"
      REAL eta = 1         UNITS "-"           "Gear efficiency (-)"
      REAL c               UNITS "mm"          "Radial play (mm)"
      REAL v               UNITS "Pa*s"        "Lubricant viscosity (Pa*s)"
      REAL l               UNITS "mm"          "Bearing length (mm)"
      REAL d               UNITS "mm"          "Bearing diameter (mm)"
      REAL W               UNITS "N"           "Bearing load (N)"
      REAL k_tor = 1       UNITS "N*m/rad"     "Spring constant (N*m/rad)"
      REAL c_tor = 1       UNITS "N*m*s/rad"   "Damping constant (N*m*s/rad)"
      REAL b = 0           UNITS "rad"         "Total backlash (rad)"
      REAL phi_rel_0 = 0   UNITS "rad"         "Initial angular distance between ports (rad)"

   TOPOLOGY
      R_GearIdeal gearRatio(ratio = ratio)
      R_Efficiency gearEfficiency (eta = eta)
      R_Hydro_Bearing hydroBearing (c = c,v = v,l = l,d = d, W = W)
      R_ElastoBacklash elastoBacklash (k_tor = k_tor,c_tor = c_tor,ddz = b,idz = -b,phi_rel_0 = 0)

   CONNECT m_in TO gearRatio.m_in
   CONNECT gearRatio.m_out TO gearEfficiency.m_in
   CONNECT gearEfficiency.m_out TO hydroBearing.m_in
   CONNECT hydroBearing.m_out TO elastoBacklash.m_in
   CONNECT elastoBacklash.m_out TO m_out

END COMPONENT
```

Fig. 4.39.: Código del componente R R_Gear_SNH en EL.

Basta con observar la forma de conectar y el orden de los componentes de la sección TOPOLOGY en el código de la Figura 4.39. y compararlo con el esquema de la Figura 4.38. para comprobar que es el mismo sistema y que se puede llegar al mismo resultado por dos caminos diferentes.

CAPÍTULO 5

5. SIMULACIÓN DE SISTEMAS MECÁNICOS ROTACIONALES

Una vez creada la librería con todos los elementos mecánicos rotacionales, es momento de empezar a simular sistemas dinámicos rotacionales a partir de estos elementos.

Para empezar a simular sistemas, es imprescindible comprobar que los elementos creados previamente reproducen de manera fiel el comportamiento de los elementos físicos que quieren representan, es decir, validar los elementos creados. Para ello, se modelarán y simularán diferentes sistemas utilizando los elementos que forman la librería rotacional mediante EcosimPro, y los resultados se compararán con los que se obtengan al simular los mismos modelos con Simulink o mediante las expresiones analíticas que los caracterizan.

5.1.- Simulación de un sistema dinámico con dos grados de libertad.

En este apartado se modelará un sistema dinámico con dos grados de libertad como el de la Figura 5.1., mediante los componentes de la librería *Rotational Library* implementada en EcosimPro.

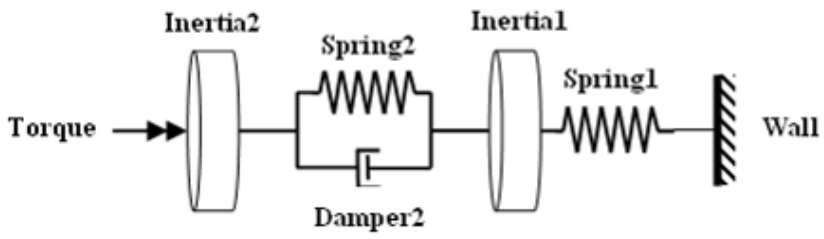


Fig. 5.1.: Croquis sistema dinámico con dos grados de libertad.

Posteriormente se simulará dicho modelo y los resultados serán comparados con los resultados obtenidos de la simulación del mismo sistema mediante la herramienta Simulink.

Simulink es una aplicación que permite construir y simular modelos de sistemas físicos y sistemas de control mediante diagramas de bloques. El comportamiento de dichos sistemas se define mediante funciones de transferencia, operaciones matemáticas, elementos de Matlab y señales predefinidas de todo tipo.

En la Figura 5.2., se puede observar el modelo del sistema con dos grados de libertad, en el entorno gráfico de EcosimPro.

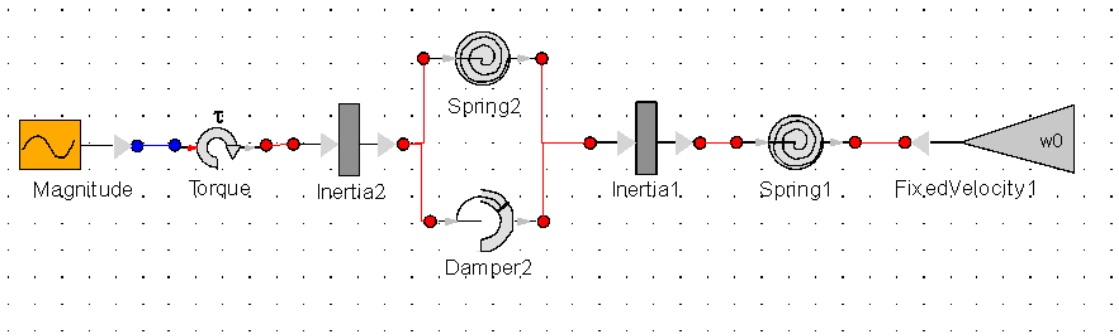


Fig. 5.2.: Croquis sistema dinámico de dos grados de libertad en Ecosimpro.

El sistema está formado por una fuente analógica de señales (*Magnitude*), un actuador de momento (*Torque*), dos inercias (*Inertia1 e Inertia2*), dos muelles torsionales (*Spring1 y Spring2*), un amortiguador torsional (*Damper2*) y una restricción de velocidad (*FixedVelocity1*).

La fuente analógica emite una señal, que el actuador de momento transforma en una par de torsión y éste es transmitido al resto del sistema. En el otro extremo, el elemento *FixedVelocity1*, permite imponer una velocidad de giro al extremo derecho del elemento *Spring1*.

Tras crear el modelo del sistema a simular, el siguiente paso consiste en instanciar todos los elementos que lo componen.

En primer lugar se instancia la fuente analógica. En este caso se quiere una función impulso de amplitud *500000 [-]* (la amplitud es adimensional, ya que las unidades de la señal emitida por la fuente analógica se determinan mediante el actuador que la acompaña, en este caso el actuador Torque) con un ancho de escalón de medio segundo y con un periodo de *100 [s]*. En la Figura 5.3. se muestran los atributos modificados de la fuente analógica.

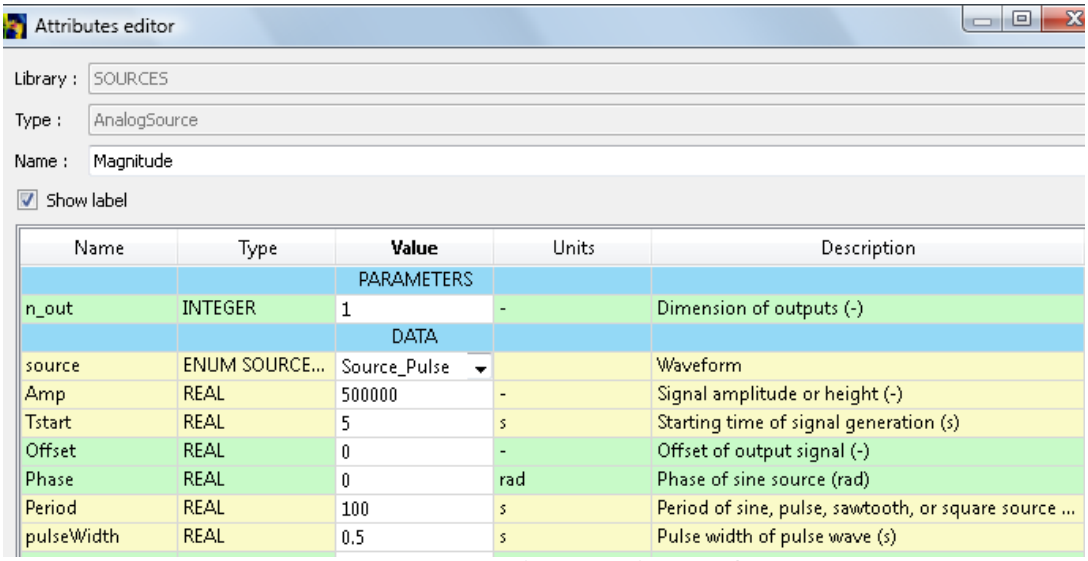


Fig. 5.3.: Atributos modificados de la fuente analógica.

Del mismo modo se procede a instanciar el resto de los elementos del sistema. En la Tabla 5.1., se representan las instancias realizadas sobre cada uno de los elementos del sistema.

Tabla 5.1.: Instanciación de los elementos del sistema de dos grados de libertad.

Elemento	Parámetro o dato	Tipo	Valor	Unidades
Inertia1/Inertia2	*I*	REAL	4000	[kg·m^2]
	phi_0	REAL	0	[rad]
Spring2/ Spring1	*phi_rel_0*	REAL	0	[rad]
	k_tor	REAL	157913.4	[N·m/rad]
	phi_natural	REAL	0	[rad]
Damper2	*phi_rel_0*	REAL	0	[rad]
	c_tor	REAL	12566.36	[N·m·s/rad]
FixedVelocity1	*w0*	REAL	0	[rad/s]

Como se puede observar en la Tabla 5.1., los elementos *Inertia1* e *Inertia2*, así como *Spring1* y *Spring2*, son equivalentes entre sí. Además, todos los elementos parten de un estado inicial, en el que la velocidad angular y la posición angular son iguales a cero.

Por otro lado, la instancia del parámetro *w0* del elemento *FixedVelocity* es nula, así se garantiza que durante todo el experimento, tanto la posición como la velocidad del extremo derecho del elemento *Spring1* serán nulos, es decir, ese punto no se desplaza, está fijo.

Los resultados obtenidos al simular el sistema de dos grados de libertad mediante EcosimPro serán comparados con los obtenidos al simular el sistema de la Figura 5.4. mediante Simulink. Es evidente que para obtener resultados comparables, las instancias de los elementos del modelo de Simulink deben de ser las mismas que las de la Figura 5.2. de EcosimPro.

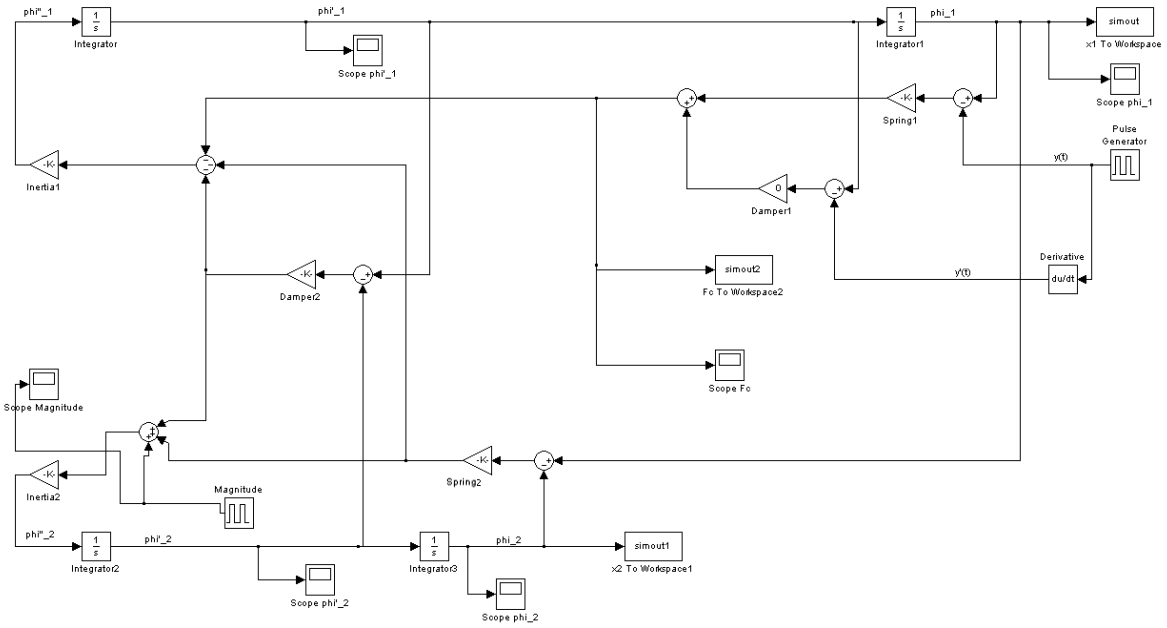


Fig. 5.4.: Croquis sistema dinámico de dos grados de libertad en Simulink.

El primer dato a comparar será el par de torsión que excita el sistema. Si las señales que excitan ambos sistemas no coinciden será difícil poder extraer conclusiones comparando ambos modelos.

En la Figura 5.5., se representa el valor de la variable *Torque.m_out.tau* frente al tiempo. Como se indicó en la Figura 5.3., se trata de un pulso de amplitud *500000 [N·m]* que actúa transcurridos *5[s]* con una duración de *0,5[s]*. Como el periodo de esta señal es mayor que la duración del experimento, el pulso sólo actúa en una ocasión.

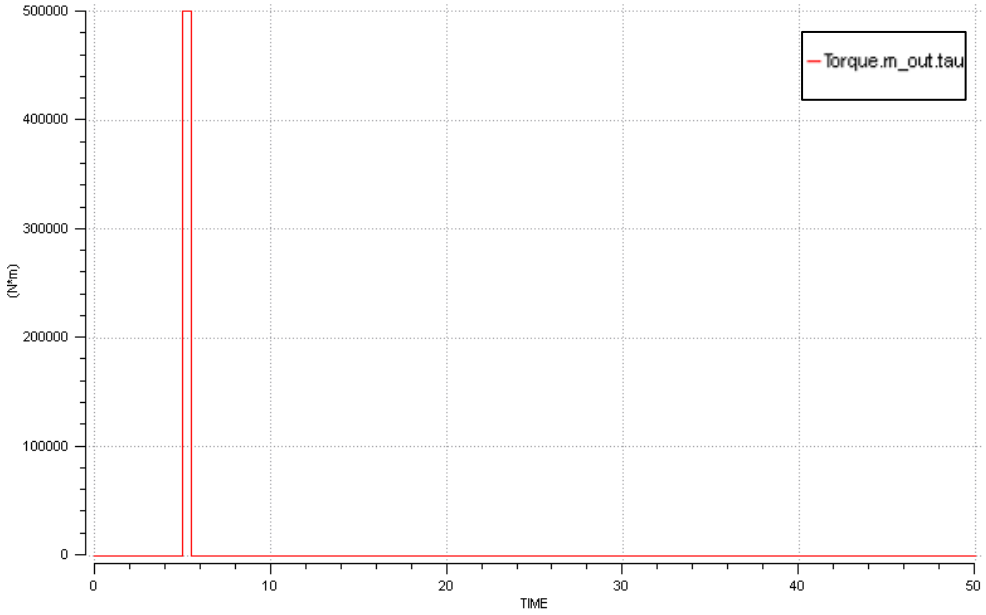


Fig. 5.5.: Representación de la variable Torque.m_out.tau frente al tiempo.

Como se observa en la Figura 5.6., el pulso que actúa en el modelo de Simulink es idéntico al de la Figura 5.5., con lo que se puede proceder a comparar los resultados de ambos modelos.

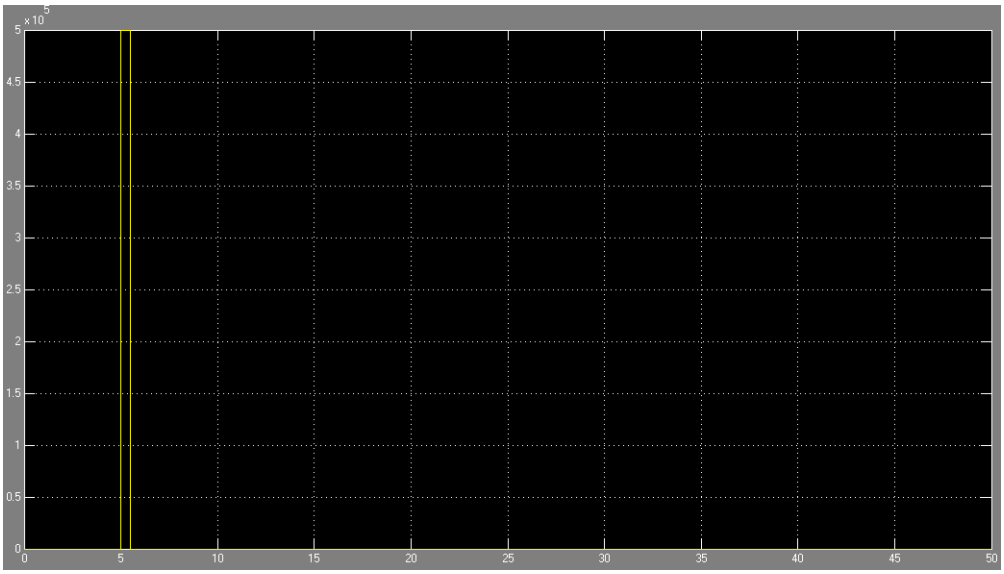


Fig. 5.6.: Representación temporal de la señal que excita el modelo de Simulink.

En primer lugar se compararán las posiciones de los dos elementos inerciales (*Inertia1* e *Inertia2*). Debido a que estos elementos pertenecen a la clase abstracta *R_Rigid*, la posición angular de los puertos de salida debe ser la misma que la de sus puertos de entrada, es decir:

$$Inertia1.m\_in.phi = Inertia1.m\_out.phi = Inertia1.phi \quad (5.1.)$$

$$Inertia2.m\_in.phi = Inertia2.m\_out.phi = Inertia2.phi \quad (5.2.)$$

Por tanto, y de acuerdo con las Ecuaciones (5.1.) y (5.2.), al representar las variables *Inertia1.m_in.phi*, *Inertia1.m_out.phi*, *Inertia1.phi*, *Inertia2.m_in.phi*, *Inertia2.m_out.phi* e *Inertia2.phi*. en un mismo gráfico en la Figura 5.7., sólo deben aparecer dos curvas, una de las cuales representa el ángulo girado por el elemento *Inertia1* y otra para el elemento *Inertia2.*

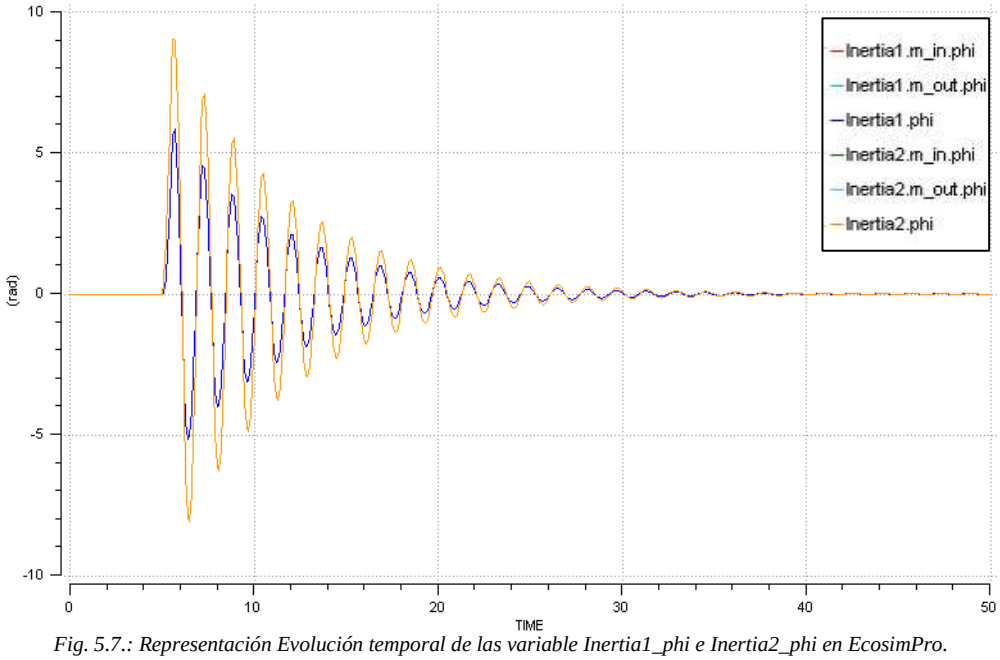


Fig. 5.7.: Representación Evolución temporal de las variable Inertia1_phi e Inertia2_phi en EcosimPro.

Los resultados obtenidos en la simulación muestran que la amplitud de las oscilaciones del elemento *Inertia2* son mayores que las del elemento *Inertia1*, lo que era de esperar al encontrarse el elemento *Inertia2* más próximo a la fuente. Además, la presencia del elemento *Damper2* da lugar una disipación de energía que reduce en el tiempo la amplitud de las oscilaciones de los elementos *Inertia1* e *Inertia2*. En la Figura 5.8. se muestran los resultados que se obtienen al simular el modelo con Simulink.

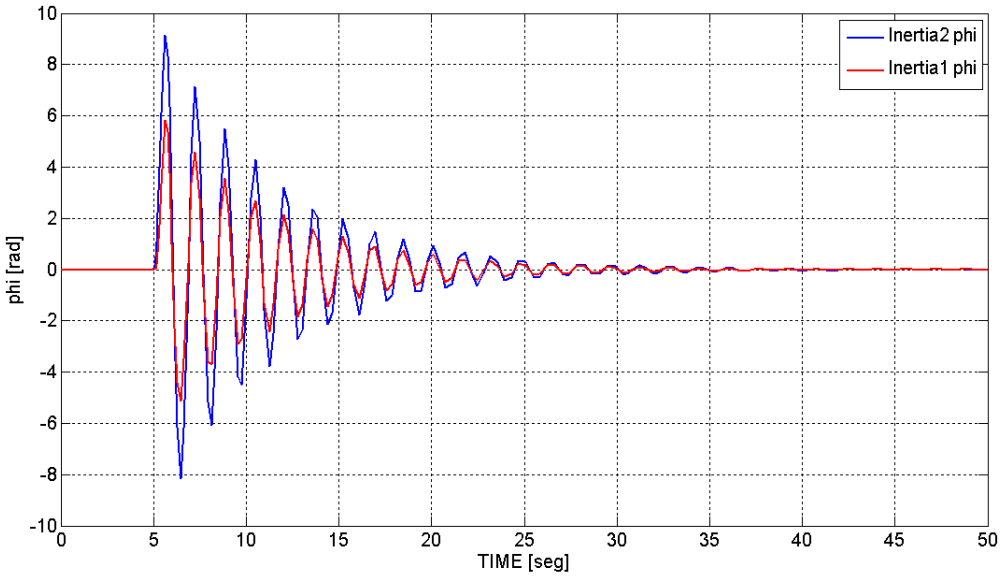


Fig. 5.8.: Evolución temporal de las variables Inertia1_phi e Inertia2_phi en Simulink.

Aunque en las Figura 5.7. y Figura 5.8., ya se observaba un comportamiento muy similar de ambos modelos, dicha similitud se hace más evidente en las Figura 5.9.a) y Figura 5.9.b), en las que se representan las curvas que unen los puntos dónde la velocidad angular de los elementos *Inertia1* e *Inertia2* cambia de signo. Un solapamiento casi total durante el experimento, confirma la validez de los modelos implementados en EcosimPro.

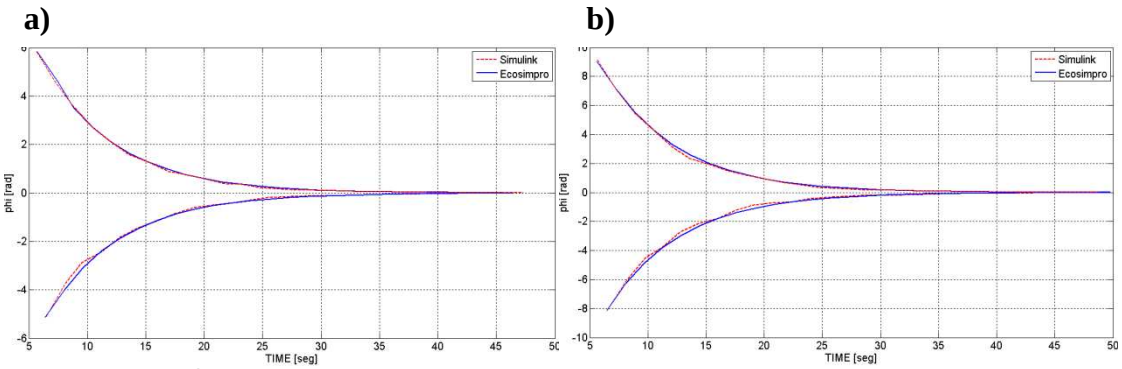


Fig. 5.9.: Evolución temporal de las posiciones angulares de cambio de signo para EcosimPro y Simulink para: a) elemento Inertia1; b) elemento Inertia2.

El siguiente paso para comprobar la validez de los modelos implementados en EcosimPro, será modificar el valor de las instancias de algunos elementos y deducir, antes de lanzar el experimento, cuál debe ser el comportamiento del modelo.

La primera instancia a modificar será la constante de amortiguamiento *c_tor*, que aparece en el elemento *Damper2* (*R_Damper* en la librería *Rotational Library*). La constante de amortiguamiento es una magnitud adimensional que mide cómo decaen las oscilaciones de un sistema después de ser perturbado, de manera que cuanto mayor es la constante de amortiguamiento, más rápido debe estabilizarse el sistema. En la Tabla 5.2. se muestran los valores de la posición angular de los elementos *Inertia1* e *Inertia2* en un instante cercano a *t=30 [seg]*.

Tabla 5.2.: Amplitud de las oscilaciones para dos valores distintos de c_tor.

		phi[rad] (*c_tor*=12566,36)	**phi[rad]** (*c_tor*=25132,27)
Inertia1	EcosimPro	0,1292	0,0102
	Simulink	0,1215	0,0128
Inertia2	EcosimPro	0,2041	0,0145
	Simulink	0,1738	0,0170

Los valores de la Tabla 5.2., junto con la Figura 5.10., en la que se representa la evolución temporal de las posiciones angulares de *Inertia1* e *Inertia2* para un valor de *c_tor* de *25132,27[-]*, confirman que un aumento de la constante *c_tor* provoca la respuesta esperada a priori, esto es, la amplitud de las oscilaciones decaen a medida que aumenta la constante de amortiguamiento. Además, la similitud entre los resultados que se obtienen con ambas herramientas, corroboran que el comportamiento del elemento *R_Damper* es el adecuado.

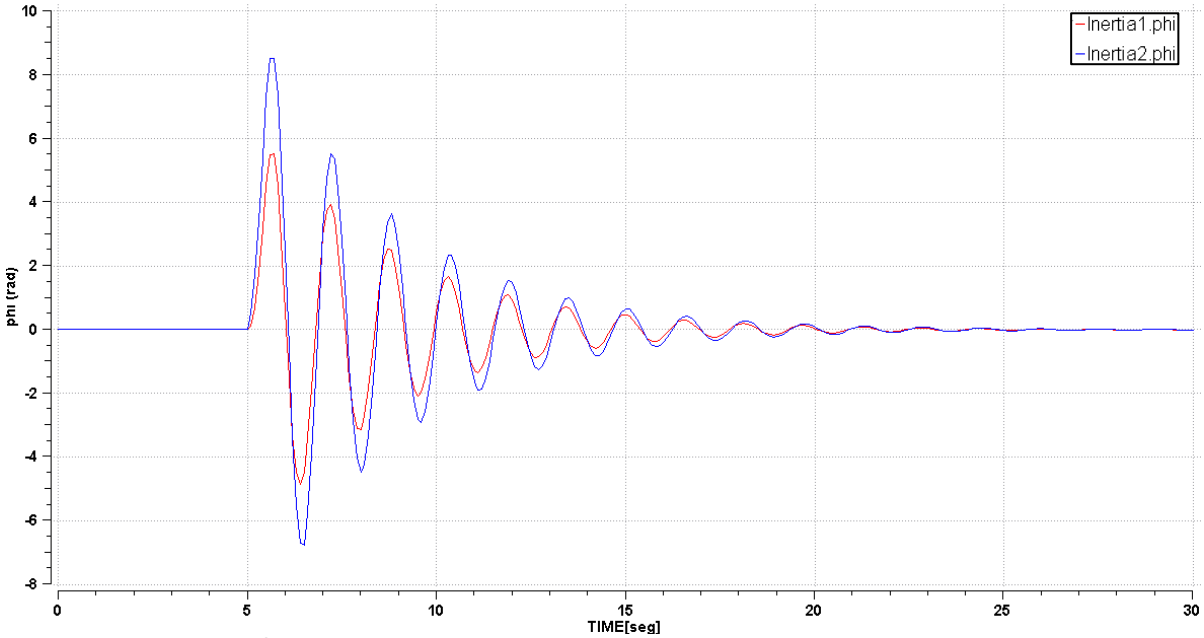


Fig. 5.10.: Evolución temporal de las variables Inertia1_phi e Inertia2_phi en EcosimPro para d=25132,27[-].

Si un aumento de la constante de amortiguamiento disminuye el tiempo que emplea el sistema en alcanzar el equilibrio, un decremento de la misma debe provocar el efecto contrario. La Figura 5.11. muestra cómo responde el sistema de dos grados de libertad para un valor unitario de *c_tor*.

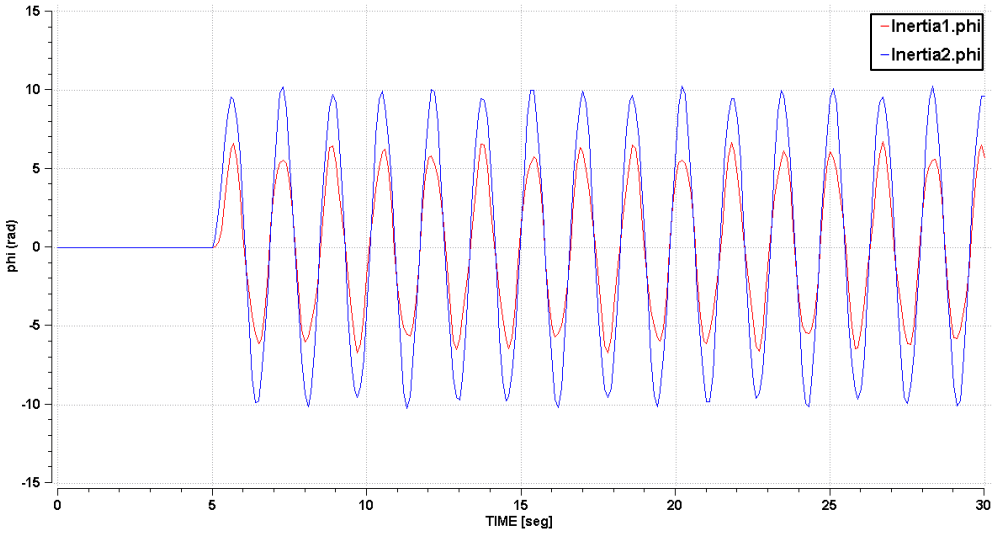


Fig. 5.11.: Evolución temporal de las variables Inertia1_phi e Inertia2_phi en EcosimPro para un valor unitario de c_tor.

En el caso de la Figura 5.11. se observa que al no existir ningún elemento que disipe la energía almacenada en los resortes, el sistema oscila contínuamente en virtud de las constantes elásticas de los elementos *R_Spring1* y *R_Spring2*. Además, cuanto mayor sea el valor de la constante elástica de los resortes torsionales, mayor será la rigidez que ofrezcan éstos y por tanto, para una misma excitación, el sistema debería disminuir la amplitud de sus oscilaciones si *k_tor* aumenta, y aumentar dicha amplitud en caso de que *k_tor* disminuya. En la Figura 5.12., se muestran distintos experimentos de los modelos de dos grados de libertad de EcosimPro y Simulink en los que se pueden

apreciar los efectos de modificar la constante elástica del *Spring2*, para un valor unitario de la constante de amortigamiento de *Damper2*.

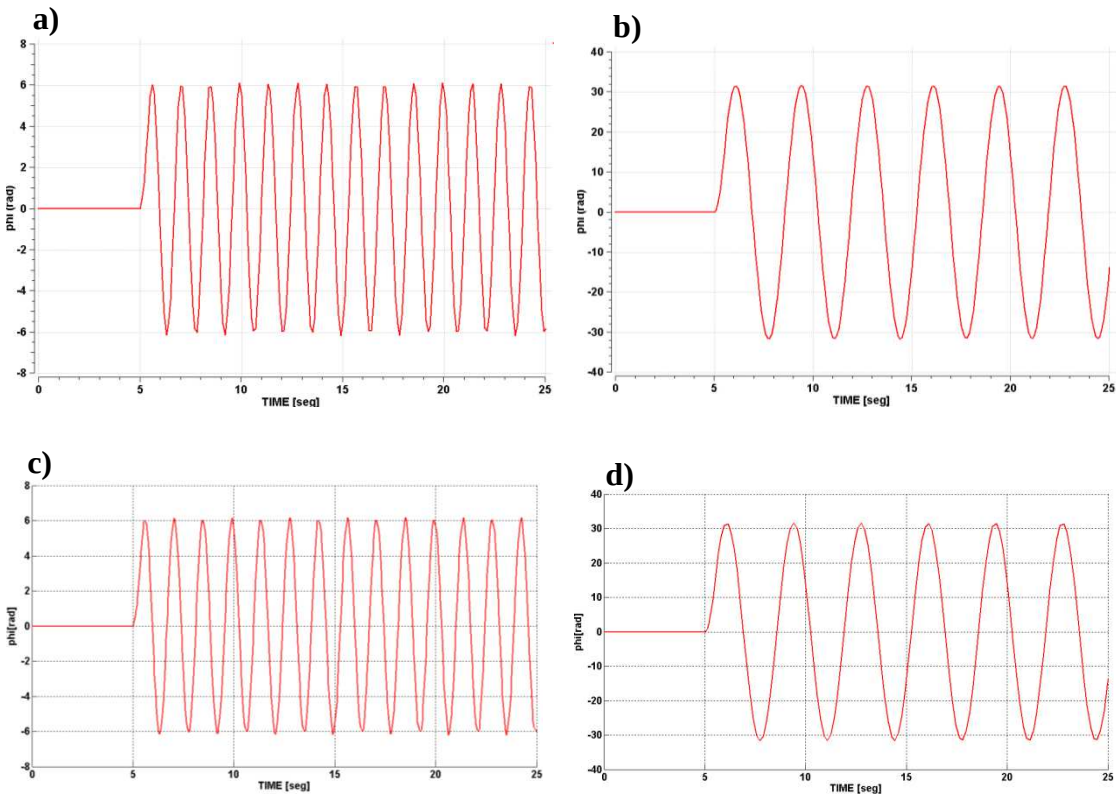


Fig. 5.12.: Evolución temporal de las posiciones angulares del elemento Inertia2 para c_tor=1 y distintos valores de la constante k_tor del elemento Spring2: a) k_tor =1579134 [Nm/rad] modelo de EcosimPro; b) k_tor =15791,34[Nm/rad] modelo de EcosimPro; c) k_tor =1579134 [Nm/rad] modelo de Simulink; d) k_tor =15791,34 [Nm/rad] modelo de Simulink.

A la vista de los resultados obtenidos en las figuras anteriores se puede afirmar que, los elementos *R_Spring* y R_*Damper* de la librería *Rotational Library* son válidos, ya que además de responder como se esperaba frente a variaciones de sus constantes características *(k_tor y c_tor)*, los resultados que se obtienen tras su simulación son prácticamente idénticos a los obtenidos con el modelo de Simulink.

5.2.- Simulación de cojinetes hidrodinámicos.

Como se explicó en la Sección 4.5.4. del capítulo anterior, un cojinete es un dispositivo mecánico que permite el movimiento relativo entre piezas en contacto, minimizando la fricción y el desgaste entre ellas. Estas ventajas implican un coste mecánico que se traduce en pérdidas energéticas, debido a efectos de cortadura en el seno del lubricante, que el componente *R_Hydro_Bearing* es capaz de determinar.

Se va a suponer un cojinete hidrodinámico con las características que se representan en la Tabla 5.3.

Tabla 5.3.: Características cojinete hidrodinámico.

Dato	Valor	Unidades
d (diámetro del muñón)	25	[mm]
l/d (relación de aspecto)	1	[-]
W (carga soportada)	1,25	[kN]
n (velocidad de giro del muñón)	1200	[rpm]
	125,66	[rad/s]
c (holgura radial)	0,02	[mm]
v (viscosidad media del lubricante)	50	[mPa·s]

Para calcular la pérdida de potencia en el cojinete de la Tabla 5.3. de forma analítica hay que seguir los siguientes pasos:

1. Calcular la carga por unidad de área proyectada según la Ecuación (4.32.), vista en el capítulo anterior:

$$P = \frac{W}{l \cdot d} = \frac{1{,}25 \cdot 10^{3}}{(25 \cdot 10^{-3})^{2}} = 2 \cdot 10^{6} \left[\frac{N}{m^{2}}\right] \tag{5.3.}$$

2. Obtener el número de Sommerfeld según la Ecuación (4.31.), vista con anterioridad:

$$S = \left(\frac{r}{c}\right)^{2} \frac{\eta N}{P} = \left(\frac{12{,}5}{0{,}02}\right)^{2} \frac{50 \cdot 10^{-3} \cdot \frac{1200}{60}}{2 \cdot 10^{6}} = 0{,}195 \ [-] \tag{5.4.}$$

3. Con el número de Sommerfeld obtenido en la Ecuación (5.4.) y la relación *l/d* dada, entrar en el gráfico de la Figura 4.28., mostrada en el capítulo anterior, para obtener la variable de fricción del cojinete, ver Fig 5.13.:

$$\frac{r}{c} \cdot f = 4{,}5 \ [-] \tag{5.5.}$$

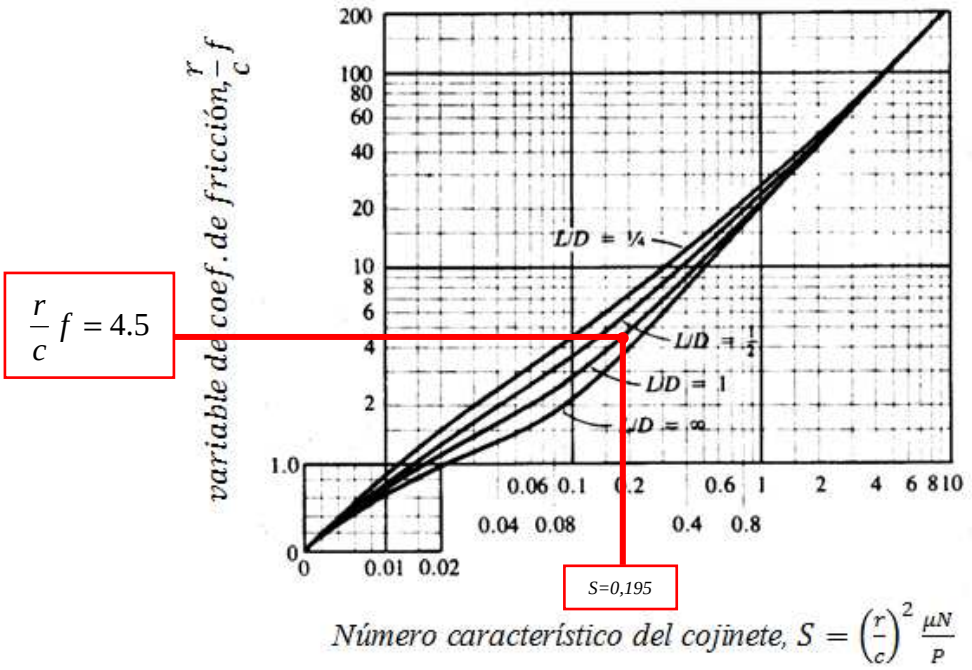


Fig. 5.13.: Variable de fricción para el cojinete del ejemplo.

4. Obtener, a partir del resultado obtenido en la Ecuación (5.5.), el coeficiente de fricción:

$$f = \frac{c}{r} \cdot 4{,}5 = 0{,}0072 \ [-] \tag{5.6.}$$

5. Por último, calcular la pérdida de potencia en el cojinete según la Ecuación (4.34.):

$$H = f \cdot W \cdot \left(n \cdot \frac{2\pi}{60}\right) \cdot r = 0{,}0072 \cdot 1{,}25 \cdot 10^3 \cdot \left(\frac{1200 \cdot 2\pi}{60}\right) \cdot \left(\frac{25 \cdot 10^{-3}}{2}\right) = 14{,}13 \ [W] \tag{5.7.}$$

Para calcular la pérdida de potencia en un cojinete como el de la Tabla 5.3. mediante el componente *R_Hydro_Bearing*, basta con simular el sistema de la Figura 5.14.

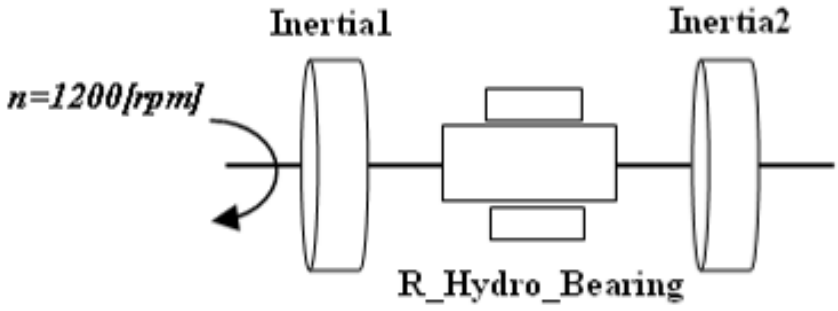


Fig. 5.14.: Croquis sistema para simular cojinete hidrodinámico.

El sistema dinámico de la Figura 5.14., que bien podría representar un eje que gira apoyado en un cojinete, consta de dos inercias (*Inertia1* e *Inertia2*) que giran a una velocidad angular de *1200 [rpm]* unidas mediante un cojinete cilíndrico con lubricación hidrodinámica (*R_Hydro_Bearing*).

El modelo en EcosimPro, en el entorno gráfico, que representa el sistema de la Figura 5.14. se muestra en la Figura 5.15.

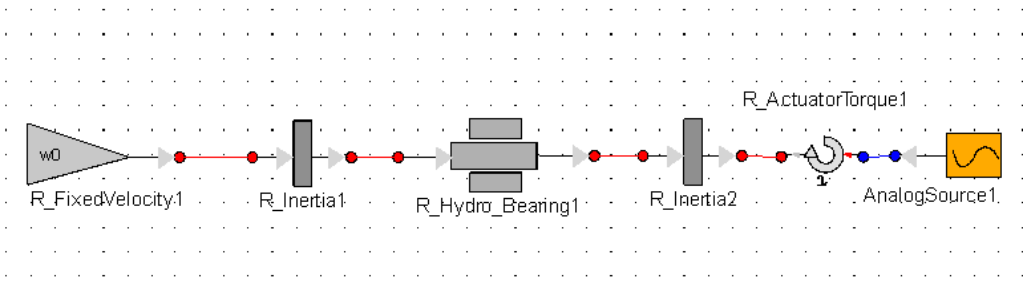


Fig. 5.15.: Modelo en EcosimPro del sistema que simula un cojinete hidrodinámico.

El elemento *R_FixedVelocity* impone una velocidad constante al sistema, en este caso se quiere que el cojinete gire a *1200 [rpm]*, es decir *125,66 [rad/s]*. La inercia de los elementos *R_Inertia1* y *R_Inertia2* no es relevante en este caso, por defecto se asigna un valor unitario. El cojinete se instancia según los datos de la Tabla 5.3., y siguiendo los criterios establecidos en la Tabla 4.51. para las unidades de las constantes introducidas por el componente *R_Hydro_Bearing*. Por último, para instanciar la fuente *AnalogSource1* basta con asignar un valor nulo a su amplitud, ya que en este caso no es necesario que esté operativa.

La Tabla 5.4. resume las instancias realizadas para los elementos que simulan el cojinete hidrodinámico.

Tabla 5.4.: Instancias para los elementos del sistema que simula un cojinete hidrodinámico.

Elemento	Parámetro o dato	Tipo	Valor
Inertia1/Inertia2	*I*	REAL	1
	phi_0	REAL	0
R_Hydro_Bearing	*c*	REAL	0,02
	v	REAL	0,05
	l	REAL	25
	d	REAL	25
	W	REAL	1250
R_FixedVelocity1	*w0*	REAL	125,66
AnalogSource1	*Amp*	REAL	0

Una vez realizadas todas las instancias se compila el modelo, se crea la partición por defecto y se genera el experimento dado en la Figura 5.16., en este caso, no es necesario modificar el código que EcosimPro genera de forma automática para el experimento.

```
EXPERIMENT exp1 ON R_Hydro_Bearing.default
   DECLS
   INIT
      -- initial values for state variables
      R_Inertia1.phi = 0

   BOUNDS
   BODY
      -- report results in file reportAll.rpt
      REPORT_TABLE("reportAll.rpt", "*")
      -- for example integrate the model 15 seconds (obtain results every 0.1 seconds)
      TIME = 0
      TSTOP = 15
      CINT = 0.1
      INTEG()
END EXPERIMENT
```

Fig. 5.16.: Código del experimento que simula el cojinete hidrodinámico.

En la Figura 5.17. se muestra la velocidad de giro del cojinete hidrodinámico durante el experimento.

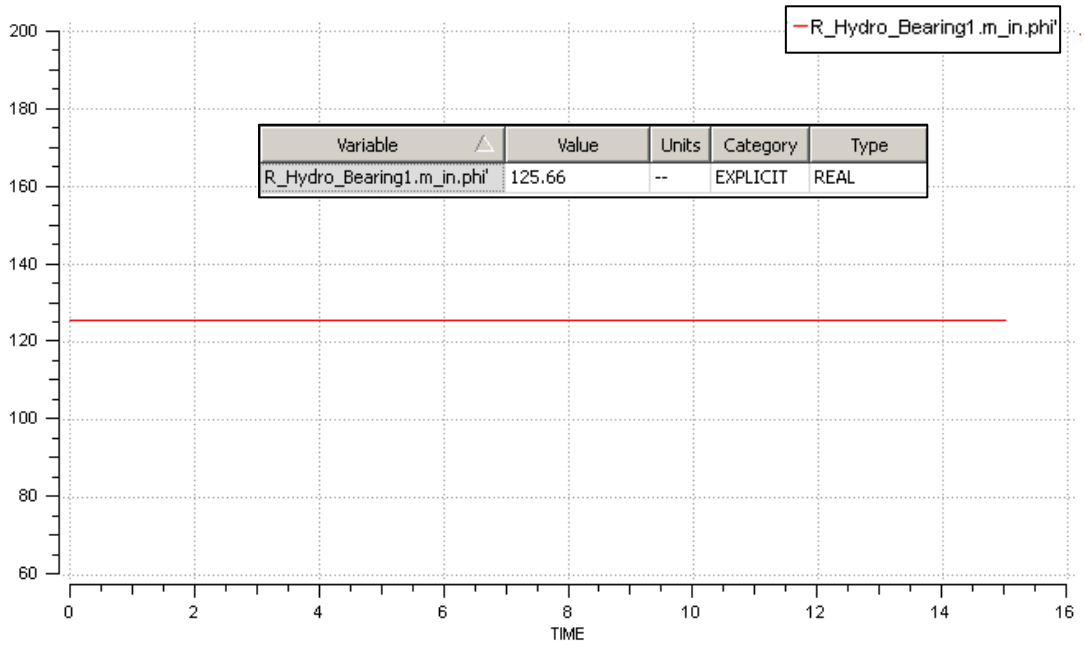


Fig. 5.17.: Velocidad de giro del cojinete hidrodinámico en [rad/s].

En la Figura 5.18. se representa la pérdida de potencia *H* que se produce en el cojinete simulado con EcosimPro. Dado que la pérdida de potencia en el cojinete obtenida en la simulación del componente *R_Hydro_Bearing* (*H=14.19 [W]*) es muy similar a la obtenida de forma analítica según la Ecuación (5.7.) (*H=14.13 [W]*), se puede afirmar que el modelo de cojinete hidrodinámico implementado en la librería *Rotational library* es válido.

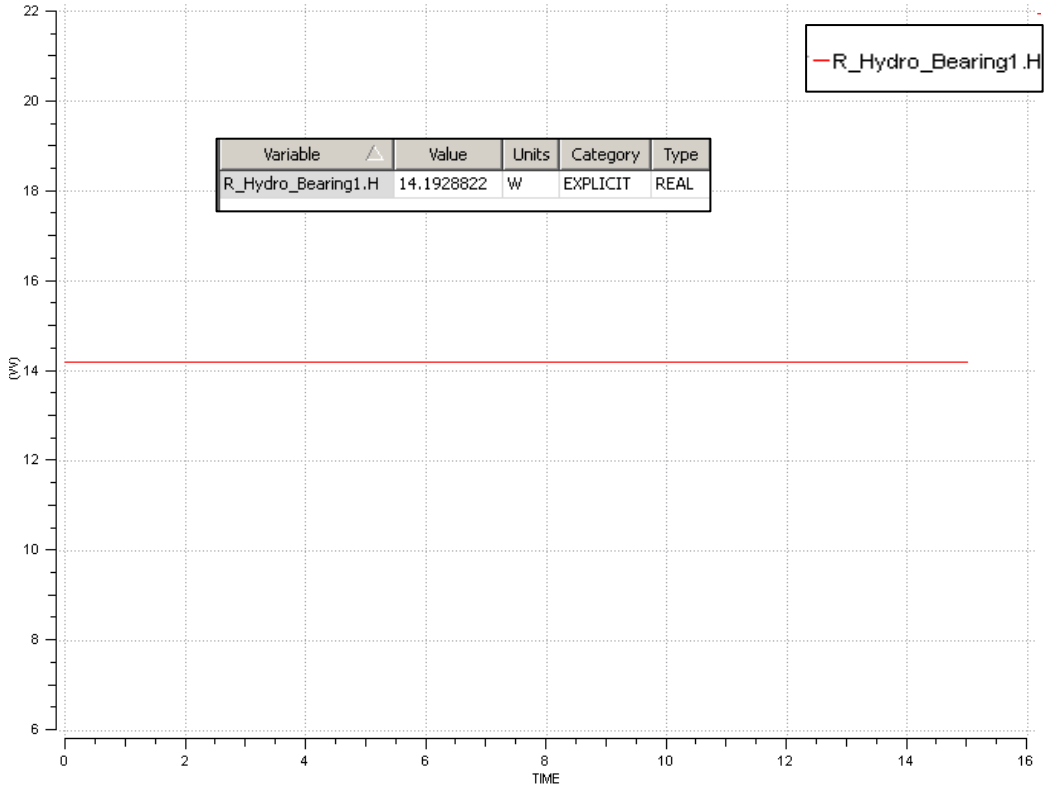


Fig. 5.18.: Pérdida de potencia en el cojinete hidrodinámico en [W].

5.3.- Simulación del cálculo de eficiencia en trenes de engranajes epicicloidales.

En la Tabla 5.5. se presenta una relación de eficiencias para diferentes tipos de engranajes. Estas eficiencias, relacionadas con las pérdidas por fricción en los dientes, son sólo para una pareja de engranajes. Si se quiere hallar el rendimiento para un tren de engranajes, o para una caja de cambios, cada uno de los pares de engranajes en la línea se deben multiplicar entre sí. Por ejemplo, dos pares de engranajes que tienen una eficiencia del 90% cada uno, si se montan en un tren de engranajes tendrían una eficiencia del 81%.

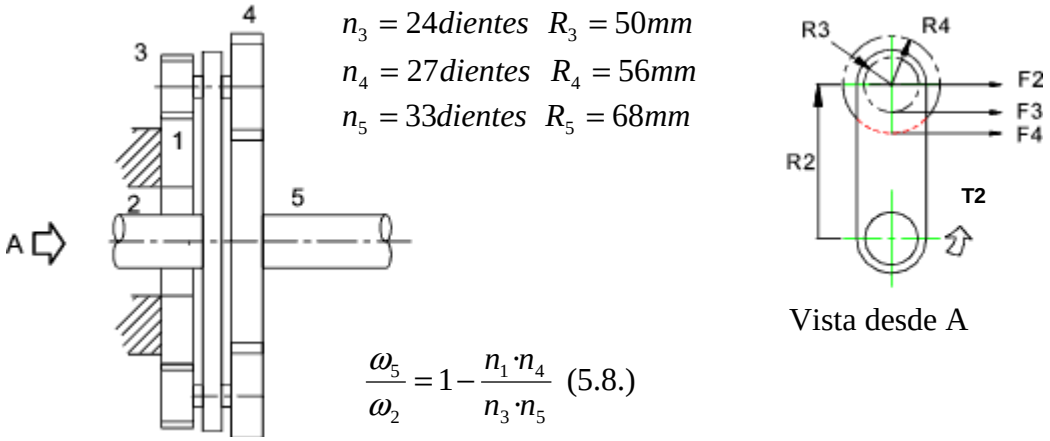
Tabla 5.5.: Tabla de eficiencias para diferentes tipos de engranajes.

Tipo de engranaje	Relación de trasmisión normal	Velocidad en la circunferencia primitiva (m/s)	Eficiencia
Cilíndrico	1:1 a 6:1	25	98-99%
Helicoidal	1:1 a 10:1	50	98-99%
Helicoidal doble	1:1 a 15:1	150	98-99%
Beveloide	1:1 a 4:1	20	98-99%
De Gusano	5:1 a 75:1	30	20-98%
Helicoidal cruzado	1:1 a 6:1	30	70-98%

Para ver cómo se hallan estos valores de rendimiento, se va a exponer una simulación para el cálculo de la eficiencia en sistemas de engranajes epicicloidales. Esta simulación

se va a resolver por un lado de manera analítica, y por otro se resolverá con la herramienta EcosimPro, creando un modelo de un componente que represente este sistema a simular.

En la simulación se considera un sistema formado por una cadena de engranajes cilíndricos epicicloidales, que se muestra en la Figura 5.19., la velocidad de entrada en el brazo 2 es $\omega_2 = 250 rpm$ y el par de entrada es $T_2 = 2{,}5 N \cdot m$, se pretende calcular el rendimiento de este sistema resolviendo las ecuaciones características de engranajes.



$$\frac{\omega_5}{\omega_2} = 1 - \frac{n_1 \cdot n_4}{n_3 \cdot n_5} \quad (5.8.)$$

$$\omega_1 = 0 rad/s \quad (5.9.)$$

Fig. 5.19.: Esquema de la cadena engranajes epicicloidales.

Con los datos dados se puede calcular la potencia que se ejerce en la entrada P_2 según la Ecuación (5.11.), a la que luego habrá que aplicar el coeficiente de rendimiento hallado en esta simulación para conocer la potencia real:

$$\omega_2 = \frac{250 \cdot 2 \cdot \pi}{60} = 26{,}18 rad/s \quad (5.10.)$$

$$P_2 = T_2 \cdot \omega_2 = 2{,}5 \cdot 26{,}18 = 65W \quad (5.11.)$$

Para calcular la fuerza en el eje 2 es necesario tener en cuenta los diámetros dados, según la Ecuación (5.12.) se sabe que:

$$F_2 = -\frac{T_2}{R_2} = -\frac{T_2}{R_1 + R_3} = -\frac{2{,}5}{0{,}075 + 0{,}050} = -20N \quad (5.12.)$$

Nótese que el signo negativo es porque las fuerzas hacia la derecha son positivas y hacia la izquierda negativas.

Mediante equilibrio de fuerzas se puede hallar las fuerzas de los engranajes 3 y 4 mediante las Ecuaciones (5.13.) y (5.14.):

$$F_2 + F_3 + F_4 = 0 \tag{5.13.}$$

$$F_3 \cdot R_3 + F_4 \cdot R_4 = 0 \tag{5.14.}$$

Resolviendo el sistema se obtienen F_4 y F_3 en las Ecuaciones (5.18.) y (5.22.) respectivamente:

$$F_3 = -F_2 - F_4 \tag{5.15.}$$

$$-(F_2 + F_4) \cdot R_3 + F_4 \cdot R_4 = 0 \tag{5.16.}$$

$$F_4 \cdot (R_4 - R_3) = F_2 \cdot R_3 \tag{5.17.}$$

$$F_4 = \frac{F_2 \cdot R_3}{R_4 - R_3} = \frac{-20 \cdot 0{,}05}{(0{,}056 - 0{,}05)} = -167N \tag{5.18.}$$

$$F_4 = -F_2 - F_3 \tag{5.19.}$$

$$-(F_2 + F_3) \cdot R_4 + F_3 \cdot R_3 = 0 \tag{5.20.}$$

$$F_3 \cdot (R_3 - R_4) = F_2 \cdot R_4 \tag{5.21.}$$

$$F_3 = \frac{F_2 \cdot R_4}{R_3 - R_4} = \frac{-20 \cdot 0{,}056}{(0{,}05 - 0{,}056)} = 187N \tag{5.22.}$$

Ahora se va a estudiar de manera aislada la primera parte del mecanismo, formada por el brazo 2 y los engranajes cilíndricos 1 y 3 que forman un sistema epicicloidal. La velocidad lineal de engrane entre dientes de los engranajes 1 y 3 v_{13} viene dada por la Ecuación (5.23.):

$$v_{13} = \omega_2 \cdot R_1 = 26{,}18 \cdot 0{,}075 = 1{,}96 m/s \tag{5.23.}$$

Siguiendo con el estudio de la primera parte del mecanismo aislado, la fuerza en 3 se corresponde con la fuerza en 2 pero con diferente signo, Ecuación (5.25.), esto ocurre por el equilibrio de fuerzas y momentos dado en la Ecuación (5.54.). Nótese que la fuerza en el engranaje 3 no es la misma que la calculada en la Ecuación (5.22.) ya que se ha aislado el sistema, sin embargo la fuerza de entrada en el brazo 2 si es la misma que la dada en la Ecuación (5.12.):

$$F_2 \cdot R_3 + F'_3 \cdot R_3 = 0 \tag{5.24.}$$

$$F'_3 = -F_2 = 20N \tag{5.25.}$$

Teniendo en cuenta esta fuerza para el engranaje 3 de manera aislada, la potencia necesaria para mover el engranaje 3 sería la expuesta en la Ecuación (5.26.), donde se multiplica la fuerza tangencial del engranaje 3 aislado por la velocidad lineal de engrane de dientes del mismo engranaje:

$$P'_3 = v_{13} \cdot F'_3 = 1{,}96 \cdot 20 = 39{,}2W \tag{5.26.}$$

Si se tiene en cuenta el valor de eficiencia dado en la Tabla 5.5. para engranajes cilíndricos, se ve que la perdida típica es entre 1% y 2%. Se va a elegir un 1%, con lo cual la perdida de potencia en este sistema epicicloidal aislado es la mostrada en la Ecuación (5.27.):

$$Pc'_{13} = \frac{P'_3}{100} \cdot 1\% = \frac{39{,}2}{100} \cdot 1 = 0{,}39W \tag{5.27.}$$

Ahora se va a calcular la velocidad angular del eje de salida del sistema completo con la Ecuación (5.28.), esta velocidad se obtiene despejando la Ecuación (5.8.) dada en la Figura 5.19.:

$$\omega_5 = \left(1 - \frac{n_1 \cdot n_4}{n_3 \cdot n_5}\right) \cdot \omega_2 = \left(1 - \frac{36 \cdot 27}{24 \cdot 33}\right) \cdot 26{,}18 = -5{,}95 rad/s \tag{5.28.}$$

La velocidad lineal de engrane de dientes entre los engranajes 4 y 5 v_{45} viene dada por la Ecuación (5.29.):

$$v_{45} = (\omega_5 - \omega_2) \cdot R_5 = (-5{,}95 - 26{,}18) \cdot 0{,}068 = -2{,}18 m/s \tag{5.29.}$$

Se ha calculado en la Ecuación (5.18.) que la fuerza tangencia del engranaje 4 es $F_4 = -167N$, con este valor y con la velocidad lineal de engrane se obtiene la potencia consumida en esta parte del sistema, dada en la Ecuación (5.30.):

$$P_4 = v_{45} \cdot F_4 = (-2{,}18) \cdot (-167) = 364W \tag{5.30.}$$

Suponiendo el 1% de perdida de potencia dado en la Tabla 5.5., se tiene que la perdida en esta parte del sistema es la dada en la Ecuación (5.31.):

$$Pc_{45} = \frac{P_4}{100} \cdot 1\% = \frac{364}{100} \cdot 1 = 3{,}64W \tag{5.31.}$$

Para acabar, se obtiene la eficiencia total del sistema de engranajes en la Ecuación (5.32.). Si al valor de la potencia en la entrada calculada en la Ecuación (5.11.) $P_2 = 65W$ le aplicamos este rendimiento, obteniendo la potencia real dada en la Ecuación (5.33.):

$$\eta = \left(1 - \frac{\left(Pc'_{13} + Pc_{45}\right)}{P_2}\right) \cdot 100 = \left(1 - \frac{(0{,}39 + 3{,}64)}{65}\right) \cdot 100 = 94\% \qquad (5.32.)$$

$$P_{2R} = P_2 \cdot \eta = \frac{65 \cdot 94\%}{100} = 61.1W \qquad (5.33.)$$

Para mostrar la flexibilidad de la herramienta de simulación EcosimPro, se va a crear un modelo de un componente que calcule el rendimiento para este tipo de sistemas de engranajes epicicloidales, de tal manera que cambiando el número de dientes, el radio de los engranajes, la velocidad de entrada o el par de entrada, se pudiera calcular el rendimiento de un sistema similar, introduciendo los nuevos datos del problema en el modelo. Este modelo no se incluirá en la librería *Rotational Library*, ya que no es un componente genérico, es simplemente una composición particular de engranajes para un determinado tipo de sistema.

El código del modelo que describe el sistema en EL es el mostrado en la Figura 5.20., nótese, como pasó en la resolución por el método numérico, que es fundamental tener en cuenta el signo de las fuerzas, se toma como referencia que las fuerzas hacia la derecha son positivas y hacia la izquierda negativas.

```
-------------------------------------------------------------------------------
-- Component R_Efficiency_Gear_Spur_SNH
-------------------------------------------------------------------------------
COMPONENT R_Efficiency_Gear_Spur_SNH
"Efficiency for spur gear system"
    DATA
      REAL n1 = 36      UNITS "teeth"      "Number of theet of gear 1 (teeth)"
      REAL n3 = 24      UNITS "teeth"      "Number of theet of gear 3 (teeth)"
      REAL n4 = 27      UNITS "teeth"      "Number of theet of gear 4 (teeth)"
      REAL n5 = 33      UNITS "teeth"      "Number of theet of gear 5 (teeth)"
      REAL R1 = 0.075   UNITS "m"          "Radius gear 1 (m)"
      REAL R3 = 0.050   UNITS "m"          "Radius gear 3 (m)"
      REAL R4 = 0.056   UNITS "m"          "Radius gear 4 (m)"
      REAL R5 = 0.068   UNITS "m"          "Radius gear 5 (m)"
      REAL wrpm2 = 250  UNITS "rpm"        "Radial velocity of gear 2 (rpm)"
      REAL T2 = 2.5     UNITS "N*m"        "Toruqe counterclockwise gear 2 (N*m)"
      REAL eta = 0.01   UNITS "-"          "Spur gear efficiency (-)"

   DECLS
      REAL w2      UNITS "rad/s"  "Radial velocity of gear 2 (rad/s)"
      REAL w5      UNITS "rad/s"  "Radial velocity of gear 5 (rad/s)"
      REAL F2      UNITS "N"      "Force gear 2 (N)"
      REAL F3      UNITS "N"      "Force gear 3 (N)"
      REAL Fi3     UNITS "N"      "Force gear 3 when insolation system gear 1 and 3 (N)"
      REAL F4      UNITS "N"      "Force gear 4 (N)"
      REAL v13     UNITS "m/s"    "Lineal velocity of tooth engagement gears 1 and 3 (m/s)"
      REAL v45     UNITS "m/s"    "Lineal velocity of tooth engagement gears 4 and 5 (m/s)"
      REAL P2      UNITS "W"      "Power gear 2 (W)"
      REAL Pi3     UNITS "W"      "Power gear 3 when insolation system gear 1 and 3 (W)"
      REAL P4      UNITS "W"      "Power gear 4 (W)"
      REAL Pc13    UNITS "W"      "Power loss between gears 1 and 3 (W)"
      REAL Pc45    UNITS "W"      "Power loss between gears 4 and 5 (W)"
      REAL etaT    UNITS "-"      "System Efficiency (-)"

   CONTINUOUS
      w2 = wrpm2*2*3.14/60
      P2 = T2*w2
      w5 = (1-(n1*n4)/(n3*n5))*w2
      F2 = -T2/(R1+R3)
      F2+F3+F4 = 0
      F3*R3+F4*R4 = 0
      v13 = w2*R1
      v45 = (w5-w2)*R5
      Fi3 = -F2
      Pi3 = v13*Fi3
      Pc13 = Pi3*eta
      P4 = v45*F4
      Pc45 = P4*eta
      etaT = (1-((Pc13+Pc45)/P2))*100

END COMPONENT
```

Fig. 5.20.: Código para sistemas de engranajes cilíndricos epicicloidales en EL.

Simulando el experimento de la Figura 5.21., se va a obtener un valor de rendimiento y varios valores de fuerzas, potencias y velocidades en los engranajes, estos resultados se van a comparar con los obtenidos numéricamente para comprobar la validez del modelo. En este caso, al ser un sistema que no varía con el tiempo, en el experimento usado se ha elegido el tiempo de simulación dado por defecto, pero se puede introducir el valor de tiempo que se desee, siempre y cuando sea positivo, para que se inicie la simulación del sistema. Otro detalle a tener en cuenta es que no se van a dar valores a los datos en el código del experimento porque en el código del modelo ya existen valores iniciales definidos, los de la simulación objeto de estudio, pero se recuerda al

lector que si se quiere simular un sistema igual con otras características diferentes, es aquí dónde hay que introducir los nuevos datos, en el bloque INIT.

```
EXPERIMENT exp1 ON R_Efficiency_Gear_Spur_SNH.default
   DECLS
   INIT
   BOUNDS
   BODY
      -- report results in file reportAll.rpt
      REPORT_TABLE("reportAll.rpt", "*")
      -- for example integrate the model 15 seconds (obtain results every 0.1 seconds)
      TIME = 0
      TSTOP = 15
      CINT = 0.1
      INTEG()
END EXPERIMENT
```

Fig. 5.21.: Experimento para obtener el valor del rendimiento del sistema en EcosimPro.

El resultado obtenido para la eficiencia total del sistema se muestra en la gráfica de la Figura 5.22., donde se representan el rendimiento total del sistema en función del tiempo, como ya se ha comentado en el apartado anterior, en este caso representar un valor en función del tiempo no es transcendente debido a que el sistema permanece invariable a lo largo de su vida. Como puede observarse, el resultado obtenido es en torno al 93,8 %, que es muy parecido al que se ha hallado analíticamente, 94 %.

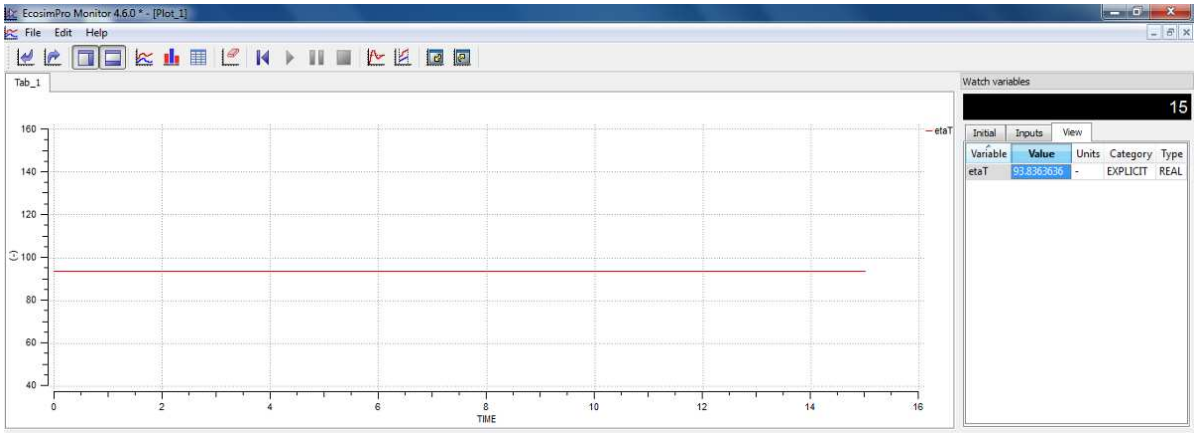


Fig. 5.22.: Gráfica para el rendimiento total del sistema.

Para comprobar que el resto de valores obtenidos (fuerzas, potencias y velocidades en engranajes) coinciden tanto si se calculan de forma analítica como si se usa la simulación en EcosimPro, se van a representar en las gráficas de las Figuras 5.23. y 5.24. estos valores.

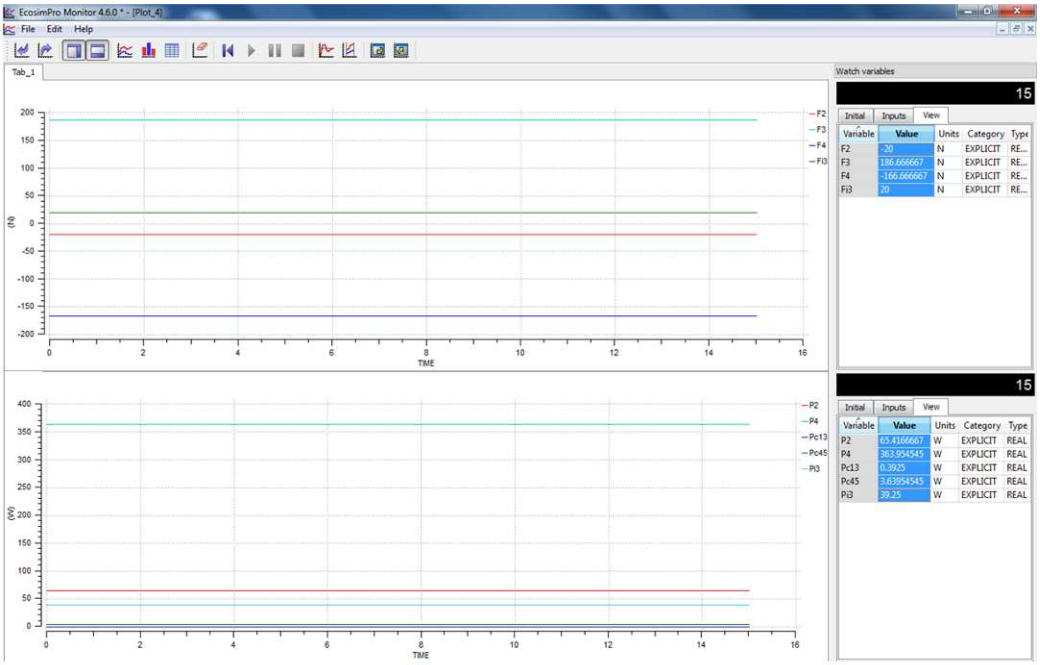


Fig. 5.23.: Gráfica para las fuerzas y potencias totales y parciales en engranajes.

En la gráfica superior de la Figura 5.23. se muestra el valor de las fuerzas totales y parciales de los engranajes del sistema, mientras que en la gráfica inferior se muestran los valores de las potencias para los engranajes del sistema completo y del sistema aislado. Se comprueba que los valores coinciden en modulo y signo con los obtenidos numéricamente.

Para acabar con la simulación, se representa en la gráfica de la Figura 5.24. la velocidad lineal entre los dientes de los engranajes y en la gráfica inferior la velocidad angular de los engranajes necesarias para hallar el rendimiento total. También coinciden ambos datos sin depender del camino tomado para llegar a ellos.

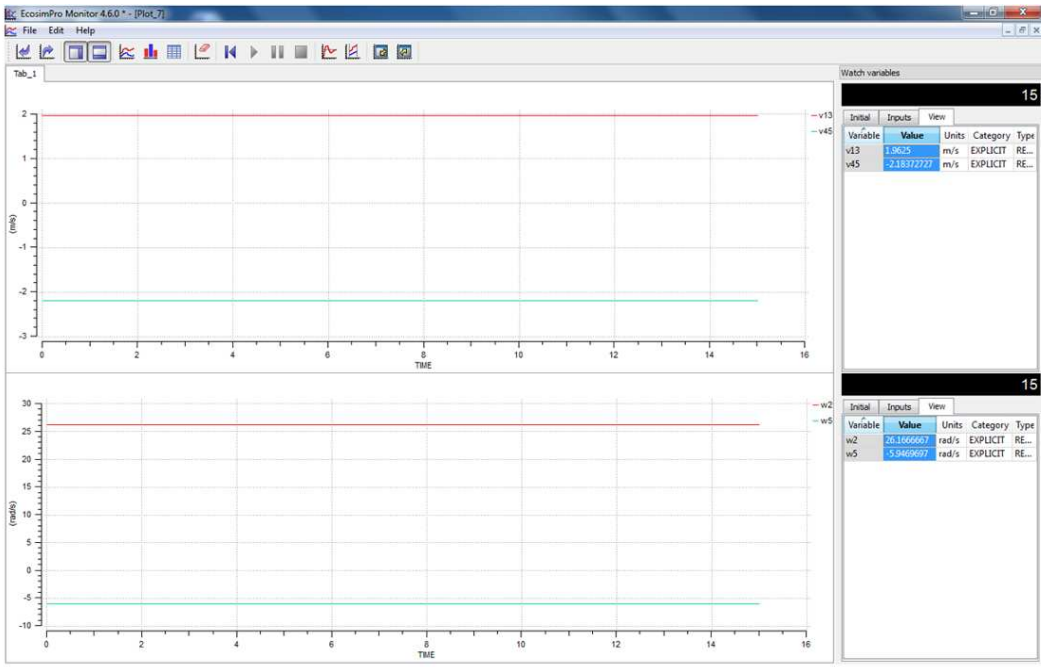


Fig. 5.24.: Gráfica para las velocidades lineales en dientes de engranajes y angulares en engranajes.

CAPÍTULO 6

6. CONCLUSIONES Y DESARROLLOS FUTUROS

6.1.- Conclusiones.

El objetivo de este proyecto ha sido la simulación de sistemas mecánicos dinámicos rotacionales mediante una herramienta de simulación informática con un lenguaje de modelado orientado a objetos. Para ello se ha desarrollado una librería que, posteriormente, ha sido utilizada para modelar y validar sistemas de dinámica rotacional.

La herramienta utilizada para tal fin ha sido EcosimPro, que constituye un software donde se recogen todas las ventajas de la programación orientada a objetos (MOO) ya descritas con anterioridad. Esta herramienta, además añade las ventajas del potente entorno gráfico del que dispone, que permite modelar multitud de sistemas en un tiempo muy reducido y de un modo sencillo. Por todo esto y tras haber trabajado durante mucho tiempo con EcosimPro se puede afirmar que es una potente herramienta, tanto a nivel docente como en el ámbito profesional, para simular cualquier tipo de sistema mecánico.

Las conclusiones sólo pueden ser positivas si se observan las gráficas obtenidas en el Capítulo 5, donde se han simulado sistemas dinámicos con dos grados de libertad, sistemas con cojinetes hidrodinámicos y trenes de engranajes. Se aprecia que tanto las gráficas obtenidas con la herramienta EcosimPro, como las obtenidas con la herramienta Matlab-Simulink o de manera analítica, tienen diferencias prácticamente despreciables, lo cual demuestra la validez de los modelos creados con la herramienta objeto de estudio.

Ha sido de especial interés para el autor del proyecto final de carrera el haber trabajado con una herramienta informática, esto le ha aportado muchos conocimientos a nivel de programación y a su vez le ha hecho ver la posibilidad de unir con un mismo fin la ingeniería informática y la ingeniería mecánica, el de simular informáticamente sistemas mecánicos de manera lógica.

6.2.- Desarrollos futuros.

Este proyecto final de carrera se ha centrado en la creación de una librería y en la simulación de elementos mecánicos rotacionales, haciendo especial hincapié en las transmisiones rígidas con la creación de modelos de sistemas compuestos por engranajes de ruedas dentadas. Es evidente que existe un amplio horizonte por explorar en lo referente a la simulación de este tipo de sistemas mecánicos, como es la simulación de los diferentes tipos de trenes de engranajes que existe en el mercado hoy en día, tal es el caso de los trenes de impulsión o las reductoras compactas.

Resulta igual de interesante el posible estudio de otros elementos mecánicos rotacionales como pueden ser los frenos, embragues, o sistemas de trasmisiones flexibles compuestos por correas, para ello sería necesario crear los modelos que representan los elementos necesarios y realizar las simulaciones oportunas.

Para acabar, otro posible futuro estudio sería el analizar el comportamiento de la herramienta EcosimPro aplicada a otro tipo de sistemas que no sean mecánicos. Se podrían crear librerías y estudiar sistemas compuestos por elementos eléctricos, elementos neumáticos o incluso elementos dentro de la disciplina de la aeronáutica y ver si los resultados de las simulaciones son igual de válidos de lo que han sido para este proyecto final de carrera.

BIBLIOGRAFÍA

[ARA06] ARACIL J., GOMEZ-ESTERN F., (2006). "*Introducción a Matlab y Simulink*".

[ATW09] ATWOOD J., (2009). "*XML: The Angle Bracket Tax*".

[BAR96] BARCELÓ J., (1996). "*Simulación de sistemas discretos*". Publicaciones de Ingeniería de Sistemas. ISDEFE.

[BEA12] BEARDMORE R., (ref. de 28 de febrero de 2012). "*Gear: Thecnical Library*". Stocks Drive Products / Sterling Instruments. United Kingdom. [En línea] Web: http://www.roymech.co.uk/Useful_Tables/Drive/Gear_Efficiency.html

[CAM00] CAMPS R., CASILLAS L. A., COSTAL D., GIBERT M., MARTIN C., PÉREZ O., (2000). "*Bases de datos*". Master de Software Libre.

[DED99] DEDES ROZAS A. A., (1999). "*Análisis de vibraciones en cajas de engranajes*". Universidad de Concepción, Departamento de Ingeniería Mecánica.

[FRI06] FRITZSON P., (2006). "*Introducción al Modelado y Simulación de Sistemas Técnicos y Físicos con Modelica*". MathModelica.

[JAC99] JACOBSON I., (1999). "The *Unified Software Development Process*". Addison-Wesley.

[JOY98] JOYANES L.,(1998). "*Programación orientada a objetos*". Mc Graw Hill.

[KEL91] KELTON W. D., LAW A. M., (1991). "*Simulation Modeling and Analysis*". McGraw Hill.

[KIK05] KIKUUWE R., TAKESUE N., SANO A., MOCHIYAMA H., FUJIMOTO H., (2005). "*Fixed-Step friction simulation: from classical Coulomb model to modern continuous models*". Touch Tech Lab Funded By Toyota, Nagoya Institute of Technology, Aichi 466-8555, Japan.

[KLE75] KLEINROCK L., (1975). "Volume I: Theory", "Volume II: Computer Applications". John Wiley.

[MOD02] MODELICA ASSOCIATION, (2002). "Modelica Language Specification". Version 2.0.

[LAM91] LAW A.M., KELTON W.D., (1991). "*Simulation Modeling & Analysis*". Second Edition, McGraw-Hill, New York.

[LAW91] LAWRENCE J.A., (1991). "*The role of JSC Engineering Simulation in the Apollo Program Simulation*". Vol.57, nº 1, 9-16.

[LOP12] LOPEZ A., (ref. de 29 de febrero de 2012). "*Lubricación*". Departamento de Ingeniería Industrial, Universidad Antonio Nebrija, Madrid. [En línea] Web: http://www.nebrija.es/~alopezro/Lubricacion.pdf

[MAT93] THE MATHWORKS INC., (1993). "*MATLAB and SIMULINK*".

[MAY11] MAYZ ACOSTA E., (2011). "*Conocimientos Básicos del Automóvil*".

[OTT99] OTTER M., ELMQUIST H., MATTSSON S. E., (1999). "*Hybrid Modeling in Modelica based on the Synchronous Data Flow Principle*". CACSD ´99, August 22-26, Hawaii, USA.

[PAL06] PALETE GAYBOR L., (2006) "*¿Qué es Viscosidad?*". Universidad Estatal Península De Santa Elena. Facultad De Ingeniería Industrial. Diseño De Maquinas.

[PAN88] PANTELIDES C. C., (1988). "*The consistent initialization of differential-algebraic system*". SIAM.

[PID92] PIDD M., (1992). "*Computer Simulation in Management Science*". John Wiley.

[RAT99] RATIONAL CORPORATION, (1999). "*Unified Modeling Language***"**.

[RUM99] RUMBAUGH J., (1999). "*The Unified Modeling Language Reference Manual***"**. Addison-Wesley.

[SAT10] SANTIAGO Y., (2010). "*Introducción de simulación de negocios*"

[SAN12] SAN ROMAN J. L., (ref. de 29 de febrero de 2012). "*Lubricación*". Material de estudio asignatura diseño de Máquinas, Open Course Ware, Universidad Carlos III de Madrid. [En línea] Web: http://ocw.uc3m.es/ingenieria-mecanica/diseño-de-maquinas/material-deestudio

[SHA88] SHANNON R. E., (1988). "*Simulación de Sistemas. Diseño, desarrollo e implementación*". Trillas, México.

[STR97] STROUSTRUP B., (1997). "*The C++ programming language*". Third Edition. Addison-Wesley.

[THE00] THEODOSSIADES S., NATSIAVAS S., (2000). "*Non-Linear Dynamics Of Gear-Pair Systems With Periodic Stiffness And Backlash*" Journal of Sound and Vibration. - 2000.

[URQ01] URQUÍA A., (2000). "*Modelado Orientado a Objetos y Simulación de Sistemas Híbridos en el ámbito del Control de Procesos Químicos"*. Facultad de Ciencias. Universidad Nacional de Educación a Distancia.

[VAZ10] VÁZQUEZ F., JIMÉNEZ J., GARRIDO J., BELMONTE A., (2010). "*Introducción al Modelado y Simulación con EcosimPro*". Universidad de Córdoba. Área de Ingeniería de Sistemas y Automática.

[VOL78] VOLTES BOU P., (1978). "*La teoría general de sistemas*". Editorial Hispano Europea.

[WEI99] WEISSTEIN E. W., (1999). "*Right-Hand Rule"*. MathWorld. A Wolfram Web Resource.